

Research on Agile Process Models in Mobile Application Maintenance

Xiaozhou Li

University of Tampere
School of Information Sciences
Computer Science
M.Sc. thesis
Supervisor: Zheyang Zhang
26.05.2013

Abstract

Together with the rapid development of mobile technology, mobile devices are not only playing a role as mobile phones in our daily lives, but a key to the world-wide social platform with versatile functionalities. Despite of tremendous profit, mobile applications, as the major providers of specific mobile services, are facing fierce competition under the background of mobile blooming. Demanding requirements from users and the constantly changing market trends are driving software companies to provide mobile applications with well-developed features and also continuously satisfactory quality. Therefore, the maintenance of mobile applications is increasingly vital to enable software companies to maintain both profits and reputation by providing constant perfection and enhancement for their products. To provide a more agile and efficient method in guiding mobile application maintenance practice is the main goal of the work in this thesis.

In this thesis, the author provides a general description of the agile process of mobile application maintenance. Details are presented concerning maintenance analysis in agile mobile application maintenance as well. Based on data collection and case studies, three maintenance models are proposed including relevant guidelines in order to provide a better way of planning agile maintenance. Other related issues are further discussed as well.

Keywords: software maintenance, agile, process, mobile application, maintenance model

Acknowledgement

I am honored to deliver my sincere gratitude to my supervisor, mentor and friend, Zheyang Zhang. This journey was a long struggle from the very beginning, together with frustration and dead ends. I sometimes doubted whether I am a master-degree student who deserves a chance to study in such wonderful place. Thanks to the great inspiration, faith, and patience of Zheyang Zhang, I was enabled to conquer the barriers and the fears inside. You made me a better myself. Thank you.

I would also like to thank Jyrki Nummenmaa and Robert Hollingsworth, who gave me great advices in thesis structure and writing. Finally, I want to thank my parents who always be there supporting me and believing in me. I love you.

Tampere, May 2013

Xiaozhou Li

Content

Research on Agile Process Models in Mobile Application Maintenance	I
Abstract	I
Acknowledgement	II
Content.....	III
1. Introduction.....	1
2. Overview.....	5
2.1. Mobile Application Development	5
2.2. Software Maintenance	8
2.3. Agile Development and User Stories Approach	13
3. Agile Process in Mobile Application Maintenance	18
3.1. User Story Types and Relevant Concepts	18
3.2. Process Overview	20
3.3. Maintenance Board as a Tool	22
4. Maintenance Analysis.....	24
4.1. Stakeholders' Focus.....	24
4.2. User Expectation	26
4.3. Business Value.....	29
4.4. Risk	30
4.5. Other Relevant Information	32
5. Mobile Application Maintenance Models	35
5.1. Emergency-Oriented Maintenance Model.....	36
5.2. Event-Oriented Maintenance Model.....	39
5.3. Constant Maintenance Model.....	42
6. Guidelines for Planning the Use of Maintenance Models.....	46
6.1. Emergency-Oriented Maintenance Model.....	46
6.2. Event-Oriented Maintenance Model.....	49
6.3. Constant Maintenance Model.....	53
6.4. Summary	57
7. Discussion	60
7.1. Switching and Combining Maintenance Models.....	60
7.2. Balance of Agility and Models.....	62
7.3. The Role of the Decision Maker	65
8. Conclusion	68
References.....	70
Appendix 1	80

1. Introduction

As a result of the rapid evolution of science and technologies, the daily life style of people has been changed dramatically, especially after the development of modern mobile devices. In the “ancient” times, the communication of people was based on telephones bound to houses. It was somehow referred to as “point-to-point”¹ communication. When mobile phones were brought into people’s lives, the way of communication was transformed into “person-to-person”² communication. Moreover, in line with the development of hardware and internet technologies, mobile devices have evolved from a simple movable telephone to a multi-functional mobile personal processor. Especially after the launch of the innovative iPhone and the launch of the Apple App Store, the role of mobile smartphones has started to become increasingly important and versatile. Smartphones currently act more like social and entertainment devices than telephone tools. According to Thurner et al. [2013], in three years mobile internet usage will surpass that of personal computers. Moreover, mobile browsing grew at a tremendous pace from 2010 to 2012, that is, an increase of 162% worldwide.

Rapid development is always connected to fierce competition, which, conversely, is also the main driving force of development. So far, together with the development of the mobile device market, competition between the iPhone, the Android phone and the Windows phone has without doubt brought the evolution of mobile devices to a new level. As a result, the number of mobile applications increased tremendously as well. By 2013, there were over 800 thousand available applications in the Apple app store³ [Mundy, 2013] while the Google Play⁴ android market has roughly the same number. Numbers of software companies were swiftly aware of the potential mobile application market as well as its promising perspective. One of the winning examples

¹ <http://www.linktionary.com/p/point2point.html>

² <http://www.thefreedictionary.com/person-to-person>

³ <http://www.apple.com/itunes/>

⁴ <https://play.google.com/store>

in mobile application development is ‘Clash of Clans’⁵, an IOS game developed by the Finnish Company Supercell Oy⁶. This free-to-play game made a sensational US\$500,000 daily income [De Vere, 2012]. In contrast, numbers of mobile applications stayed at the bottom of the app store ranking and made limited profits or even suffered from losses. In addition to the reasons of marketing and promoting, the factors in terms of software development aspects should be more concentrated on.

One crucial part of mobile application development is the application maintenance. Similar to software maintenance in general, mobile application maintenance is both time and money consuming. There are huge numbers of new applications emerging every single day, many of which have similar functionalities. Thus, the conflict between the daily growth of users’ requirements for high quality software applications and the limitations of funds, time and experience of software project teams is becoming increasingly fierce. To pursue constantly high software quality, a proper efficient software maintenance process is the key. However, the cost of maintenance is extremely high in terms of both effort and finance.

As a result of the inefficiency and inflexibility of traditional software maintenance practices, a decently efficient process of software maintenance is mandatory, especially for mobile application maintenance, in which constant updates are delivered dealing with constantly changing user requirements. With regard to software development process methods, the traditional waterfall model, whose essence is that complex software systems can be built in a sequential and phase-wise manner [Szalvay, 2004], had prevailed for a long time as the natural principles according to process led to the original thought of a linear model. Dealing with the drawbacks in the waterfall model, other process models were invented, e.g. Spiral model⁷, RUP⁸, iterative method⁹, and so on. Nevertheless, those process models contain defects as well. To cope with the defects of the predictive and

⁵ <https://itunes.apple.com/us/app/clash-of-clans/id529479190?mt=8>

⁶ <http://www.supercell.net/>

⁷ <http://dl.acm.org/citation.cfm?doid=12944.12948>

⁸ <http://www-01.ibm.com/software/rational/rup/>

⁹ <http://c2.com/cgi/wiki?IterativeDevelopment>

process-oriented software processes and avoid software failures, Kent Beck and 16 other developers co-created the “manifesto for Agile Software Development” [Beck et al., 2001] and started the era of agile method in software development.

The success of agile methods in software development has been widely acknowledged. However, although issues concerning the software maintenance process have been widely studied, studies of applying agile principles in software maintenance process are very limited. For example, in the study of Mello [2012] and Svenssen et al., [2005], agile principles were introduced into maintenance phase. However, the proposed guidelines are complex and not specified for mobile application maintenance. The aim of this thesis is to propose a concise agile development process for the mobile application maintenance phase. How to keep maintaining software with minimum effort and a guarantee of maximum profit will be the key research question in this paper.

To conduct this study, a literature review in relevant fields was firstly performed. Studying articles and scientific papers concerning relevant focused issues to a large extent provides a better perspective in the target direction. Based on the literature research, the idea of innovation of an agile maintenance process model for mobile application maintenance is conceived, considering the defects of traditional software maintenance process. Therefore, an explicit concise agile maintenance process is constructed as the core answer to the research question. Thus, compared to the structure of agile methods in software development, an agile process for mobile application maintenance is constructed.

To better describe the agile maintenance process, general guidelines for the process are required. On the base of the analysis of a number of real-life application cases, the fact is that different mobile applications deliver updates with different contents, tendencies and paces. Thus, studies of real-life mobile application cases and analysis on relevant influential factors is conducted to find regular patterns of maintenance updates in terms of update timeline and content. Based on the conceived regulations, in this study, three maintenance models are constructed and summarized

in order to further clarify the practices of the presented maintenance process. Additionally, based on a further analysis on certain special application cases from each model category, more guidelines for each mobile application maintenance model are proposed as well. One of the most important aspects covered is the guidelines for prioritizing different types of user stories for the adoption of each maintenance model. Therefore, by following the guidelines of the mobile maintenance models, software companies are able to perform maintenance practices in an agile and efficient way.

The mobile application cases were selected from the IOS platform, which is so far the most popular mobile platform with the most available applications. In addition, the update information is much easier to collect [Sheridan, 2012] compared to Android OS. The application market regions chosen are the USA and China. The application store from the USA region contains by far the most available applications, while China has the most rapidly increasing number of applications on the Apple app store [Russell, 2012][Koekkoek, 2011][Hughes, 2012]. Moreover, the fact that the two countries are distinguished from each other in terms of customs and culture in many ways makes it easier and more credible that a universal set of maintenance models could be adopted beyond objective circumstances. All the possible categories listed in the Apple app store will be covered.

In this thesis, there are eight chapters including this introduction chapter. In Chapter 2, a brief overview of background information about mobile application, software maintenance, and agile development method will be presented. In Chapter 3, the discussion will concentrate on the agile process model for mobile application maintenance and relevant concepts. Chapter 4 will provide guidelines for the maintenance analysis phase in mobile application maintenance process. Chapter 5 will introduce the mobile application maintenance models, while guidelines in details will be discussed in Chapter 6. In Chapter 7 a further discussion concerning the maintenance models will be presented. At last, Chapter 8 will be a brief conclusion including the limitation of this thesis and relevant future work.

2. Overview

In this chapter, general background information is presented. The background information covers general information concerning mobile applications, software maintenance and agile development methods. The overview aims to show the fields that this study will discuss as well as the motivation of the study.

2.1. Mobile Application Development

Mobile devices are playing an increasingly important part in people's daily life. With the main driving force of innovative progress of functionalities and development in hardware performance, mobile devices have evolved to be a versatile device providing various services in order to satisfy people's requirements for business, entertainment, social, and so on. As the software products that are designed to run on those mobile devices, mobile applications are the main mobile functionality providers. It all began from the first commercially available mobile phone ever, which was called "the brick", Motorola DynaTAC 8000X¹⁰. It was firstly marketed in 1983, with the weight of 2.5 pound and price of almost US\$4,000, providing phone call features and an application of contacts. At that time, developing applications for mobile phones was the inside job performed by manufacturers which made third-party applications unavailable.

Through all these years, mobile applications went through a long process of evolution, from the Wireless Application Protocol (WAP) [Paukkunen, 1999] standard with low speed and high data transferring cost, to the rise of Personal Digital Assistant (PDA) [Viken, 2009], to the variety of platform competition between Palm OS¹¹, Sun Microsystems¹² with J2ME¹³, BREW¹⁴, and Symbian OS¹⁵,

¹⁰ <http://www.retrobrick.com/moto8000.html>

¹¹ <http://www.palminfocenter.com/palm-os/>

¹² <http://sun-microsystems-inc.software.informer.com/>

¹³ http://java.com/en/download/faq/whatis_j2me.xml

¹⁴ <https://www.brewmp.com/>

until now the blossom of IOS and Android.

On July 10th, 2008, Apple App Store was integrated with iTunes via an update. On the very next day, the iPhone 3G was released with a pre-installed IOS 2.0.1, on which users were able to launch the App Store. Even earlier, Software Development Kit for iPhone OS was released, which enabled developers to develop their own mobile applications on MacOS¹⁶ 10.5.4 or higher with Xcode¹⁷. With the launch of the Apple App Store and the great success of iPhone, smartphone entered the era of touch screen, which changed the whole model of mobile application market. Users were enabled to use Wi-Fi or cellular network for installation without a personal computer. As a competitor, Google launched Android Market on October 22nd, 2008 as well. Within these years, both the mobile application market and the developer community were split into these two major camps.

There were 500 applications on Apple App Store on the first day when there were already 10,000,000 total downloads 3 days later. Till now, according to the statistics in January 2013, the total download from Apple's App Store added up to 40 billion [Miller, 2013]. From 2010 to 2012, there was a tremendous growth in mobile applications, especially for IOS and Android platform [MobiThinking, 2012]. Not only just for iPhone users does the huge boom occur, but the trend is also global and bigger than ever [Erica Ogg, 2011]. The total download of Apple app stores was added up to 40 billion of which 20 billion happened in 2012 [Reisinger, 2013] with also approximately 7 million song downloads from iTunes per day [Dong, 2013]. According to the report of Don Reisinger, there are now over 500 million active accounts in Apple app store and 775,000 applications [Reisinger, 2013]. Additionally, in 2013, the total downloads worldwide of mobile applications are predicted to be around 165 billion compared with the number of 82 billion in the previous year [MoWeble, 2013] when the revenue will increase by 62% up to US\$25 billion as well. In addition, IOS and Android platforms also attract the interests from 80% of

¹⁵ http://www.developer.nokia.com/Community/Wiki/Symbian_OS

¹⁶ <http://www.apple.com/osx/what-is/>

¹⁷ <https://developer.apple.com/xcode/>

both enterprises and developers in application development [Smith, 2013] [XinhuaNet, 2013].

Mobile application market is under an incredibly high-speed development pace and also under highly unpredictable circumstance. There are newly released application products from market regions all over the world every single day. Even within the same categories of applications, there are numbers of mobile applications sharing similar functionalities, competing for the same niche of market. Additionally, there is still great possibility that exciting new applications emerge to grab the interests of the majority of users, even in a saturated market. Therefore, dealing with the rapid development pace and unpredictability of mobile application market, an efficient development method and constant high quality guarantee is the key solution for all software companies.

According to Abrahamsson et al. [2012], mobile application development is also facing the challenges from both technical constraints as well as specific requirements including:

- Limited capabilities and rapid evolution of terminal devices;
- Various standards, protocols and network technologies;
- Need to operate on a variety of different platforms;
- Specific needs of mobile terminal users;
- Strict time to market requirements;

Thus to overcome the challenges in mobile application development, a better and more efficient method is one of the most important solutions. Agile development method is one of the best options to adopt. Agile methods provide adaptability of enterprises to a dynamic environment [Salo, 2006]. The main advantage of agile development includes concentration on business value, adaptability, customer involvement, and the frequent deliveries, which is abundantly discussed in many articles and literatures notwithstanding opposing arguments concerning agile methods based on lacking of scientific validation, difficulty in integrating plan-based practices, and uncertainty in distinguishing agile methods from ad-hoc programming

[Salo, 2006] [Boehm, 2002].

On the other hand, compared to computer based software application, mobile development has certain distinguishing attributes that are more suitable for adopting agile methods. According to Abrahamsson [2005], agile methods characteristics are suitable for mobile development in almost all aspects including high environment volatility, small development team, object-oriented development environment, non-safety critical software, application level software, small systems, and short development cycles. Additionally, Robert Holler considered agile methods as a natural fit for mobile development, as agile development continuously focuses on guaranteeing the balance between business and technology [Holler, 2006]. When agile principles are adopted, users are continuously involved to enhance the functionality and usefulness of the software product. Continuous delivery of software also ensures the capability of the team to develop and maintain software products confidently dealing with unpredictable environment.

So far there is already one agile development methodology specifically designed for mobile development, the Mobile-D agile approach presented by Abrahamsson et al. [2004] and improvement by Spataru [2010]. Briefly, the Mobile-D method encompasses nine main elements including phasing and placing, architecture line, mobile test-driven development, continuous integration, pair programming, metrics, agile software process improvement, off-site customer, and user-centered focus and five phases within the development cycle including explore, initialize, productionize, stabilize, and system test & fix.

2.2. Software Maintenance

The software maintenance phase covers the whole procedure ranging from the delivery of the software to its retirement. The definition of software maintenance was given by Martin and McGlure as follows, “Changes that have to be made to computer programs after they have been delivered to customer or user” [Martin et al., 1983]. A further definition is given by IEEE [1998] “the modification of a software

product after delivery to correct faults, to improve performance or other attributes, or to adapt the product to a modified environment”. In the following sections, the introduction will cover the details of software maintenance in terms of significance, types and process respectively.

2.2.1. Significance of Maintenance in Software Life Cycle

Software maintenance has played an increasingly important part in the development and evolution of software engineering theory and practice. No single team is able to produce perfect software satisfying all requirements and being bug-free, thus maintenance is inevitable. Even though there might be some software that works well, there are still changes and emerging new ideas that would enhance the functionalities of the software. Therefore, software maintenance is and will for a long time be one of the most important phases in software engineering.

However, software maintenance is also expensive and time-consuming as well. The duration of software products normally ranges from several years to decades, during which maintenance is always required. In 2000, Van Vliet [2000] claimed that 100 billion lines of code are produced in the world within which as much as 80% is poorly structured or documented. After the releases of software products, undercover bugs which still remain unrevealed will be reported by users, which make the maintenance difficult to perform, especially when software is poorly structured. The cost of maintenance has increased by incredible amounts within software development. The Y2K reprogramming cost the UK \$50 billion [Neumann 1997], while at the company level Nokia Inc. used about \$90 million for preventive Y2K bug correction [Koskinen, 2003]. Additionally, the maintenance cost normally takes up a percentage ranging from 49% to 75% of the total cost of the software life cycle, which underlines the significance of software maintenance. Therefore, an efficient software maintenance process model and constructive guidelines are highly required for current software projects. The maintenance phase should cover the whole period after product release until the retirement.

Same as other products, software and the services provided are meant to be sold

to customers as well. Therefore, cost and quality are always the two prime issues to concern, during the process of manufacturing. Simply put, by reducing the cost of software development and ensuring the quality, the value of software products will be most likely maximized. According to statistics [Sehlhorst, 2007], over 90% of the cost of software development was resulting from maintenance, which is a tremendous amount of expense. Thus, a good management of software maintenance will help to reduce the cost emerging from it and to better guarantee the quality as well.

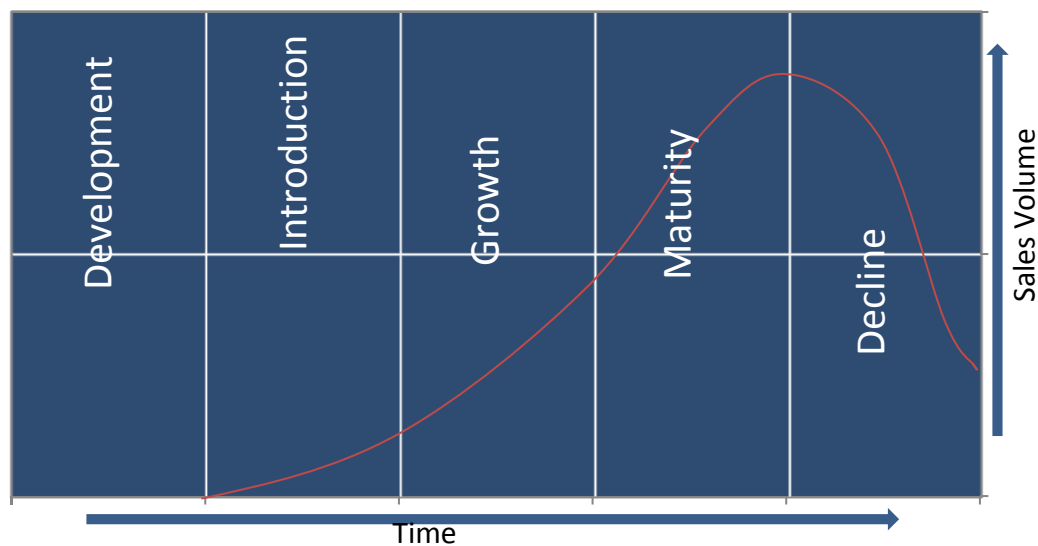


Figure 2.1 Product life cycle Diagram¹⁸

The product life cycle is a description of presence or behavior of a product in the market over time. Figure 2.1 shows the function of sale volumes and time within the product life cycle. The products are manufactured and introduced to the market followed by a growth in market, maturity and decline, which is the same with software product. However, software maintenance aims to produce constant boost in the sales volume of a software product ranging from the introduction period till the decline. As a result of the effects of maintenance, software product remains in the maturity and growth phase longer to create more sales.

¹⁸ <http://www.quickmba.com/marketing/product/lifecycle/>

2.2.2. Software Maintenance Types

Software maintenance encompasses four different types, including corrective maintenance, perfective maintenance, adaptive maintenance and preventive maintenance [Newton, 2010].

- **Corrective Maintenance**

Corrective maintenance refers to the fixing actions performed when software failures occur, that is, the bugs or relevant problems that resulting in critical failures of software products.

- **Perfective Maintenance**

Perfective maintenance aims for the perfection of software products concerning some issues influencing the usability of users which most users would prefer getting them improved but could endure being without them.

- **Adaptive Maintenance**

Adaptive maintenance deals with the issues concerning the upgrade of system environment or hardware environment.

- **Preventive Maintenance**

Preventive maintenance is certain modification on the source code in order to prevent a software system from future crash, which aims for enhancement of the maintainability and credibility.

2.2.3. Traditional Software Maintenance Process

According to IEEE standard 1219 [1998], in a traditional software maintenance process there are seven typical phases of activities: Identification, Analysis, Design, Implementation, System test, Acceptance test, and Delivery.

- **Identification**

At this point of maintenance phase, when a modification request is received, the maintenance team will firstly identify, classify and estimate it. The actual activities include assigning an identification number, classifying the maintenance type, the preliminary estimating modification size and magnitude, etc.

- **Analysis**

The analysis phase is to use the information and validated modification requests obtained in the identification phase and study the feasibility and scope of those modification requests. Subsequently, a plan for the future phases, such as design, implementation, test and delivery, will be devised. As a result, a feasibility analysis and a detailed analysis should be made.

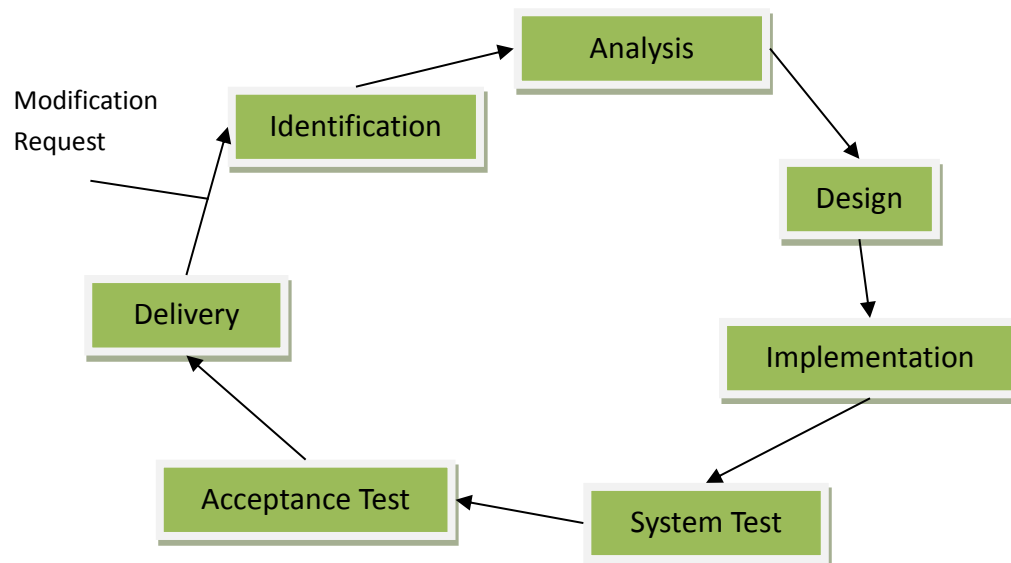


Figure 2.2 Traditional Software Maintenance Process [IEEE, 1998]

- Design

In the design phase, existing documentation and the current system will be used as well as the analysis documentation from previous phases, when actual design of modification will be made.

- Implementation

As a result of the previous phases, in the implementation phase current source code and maintenance planning documentations will be used to drive the implementation. By applying coding, unit testing, integrating, risk analyzing etc. the team should have the updated software as well as the updated documentations after this phase.

- System Test

After implementing updates to the software, a system test should be performed in order to validate the updated code. The team should make sure in system test phase

that no new bugs and faults are introduced in the previous implementation phase. Within this phase, activities such as system functional test, interface testing, and regression testing should be included.

- Acceptance Test

With the fully integrated system implemented in previous phases, an acceptance test should be performed by the customer.

- Delivery

When passing system test and acceptance test, with all necessary documentations ready, the modified system is ready for delivery.

The traditional software maintenance is possible to be put simply just like what is described previously. However, as a part of traditional software development, traditional maintenance phase is combined with various documentations. Additionally, the process described in [IEEE, 1998] is to a large extent dramatically detailed. Concerning every single phase within the process, there are explicit inputs, outputs, process description and control presented, which provides a direct guide for maintenance team. Compared to the traditional software development process, taking the example of the waterfall model, well-structured process and prescribed behaviors were considered to be predictable, which most likely results in positive consequences. However, the waterfall model derived certain inevitable defects that would cause software failures. Traditional software maintenance process shares certain defects as well.

2.3. Agile Development and User Stories Approach

2.3.1. Agile Development

Agile development is an iterative and incremental development method with a core of people who form self-organizing and cross-functional teams collaborating to assist evolution of requirements and solutions. Based on the recollection of Gerald M. Weinberg who worked in a project named Mercury running half-day time-boxed iterations, incremental development had already begun to be utilized as early as 1957

[Larman, 2003]. In 1974, an adaptive software development process was introduced by E.A Edmonds [Edmonds 1974]. Subsequently in the 90s, the term “lightweight software development method” was introduced as a reaction to the heavyweight method which was criticized as being regimented and regulated as the waterfall method. At that time, some early implementations of lightweight methods were introduced such as Feature Driven Development [Coad et al., 1999], Dynamic System Development Method [Stapleton, 1997], Extreme Programming (XP) [Beck, 2000], Scrum¹⁹. Following that, in 2001, at the Snowbird resort in Utah, 17 software developers discussed lightweight development methods and published the manifesto for Agile Software Development, as follows²⁰:

We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

Individuals and interactions over processes and tools

Working software over comprehensive documentation

Customer collaboration over contract negotiation

Responding to change over following a plan

That is, while there is value in the items on the right, we value the items on the left more.

Compared to the traditional development models which involve long-term planning throughout the whole process, agile development methods break the whole project into small increments, each of which contains plans for short term. Each increment, which is also an iteration, is time-boxed into a period from one to four weeks. The team performs as a self-organized team with cross-functional members and demonstrates a working product - normally a product with limited working functions - to the clients and possible other stakeholders. The risk of failures is minimized by quickly adapting changes within the project. More features and perfecting of some existing features will be added to the running product to constantly meet the needs of clients. Additionally, agile methods also emphasize the importance

¹⁹ <http://www.scrum.org/Scrum-Guides>

²⁰ <http://agilemanifesto.org/>

of face-to-face communication when working in the same location.

2.3.2. User Story Approach

As an efficient approach widely adopted in agile development process, user stories are usually adopted in all possible agile development frameworks including XP, Scrum and Kanban²¹. According to Mike Cohn [2004], a formal template to adopt in writing user story is

As a **ROLE**, I want **SOMETHING**, so that **BENEFIT**

In this template, the 'Role' represents the possible user of the software application, which is already categorized into certain user groups. 'Something' here in the template represents the feature which is desired by the customers or the issue needed to be solved. It can also be represented by an act that would possibly be performed by the future users and which is supposed to be explicit and small. Additionally, the 'Benefits' part is optional. To add benefits in a user story is to enhance the importance of it. Furthermore, it is also an explanation to the developers why customers want this feature done and how badly customers want it done, which would help to estimate the priority in a general sense.

According to what Mike Cohn [2004] mentioned, there are 6 attributes focused for good user stories, including Independent, Negotiable, Valuable to users, Estimatable, Small, and Testable. Within these six characters, Negotiable and Estimatable are not mentioned in good characteristics for user requirements by Karlsson [2009]. User stories are not contracts, which should be insisted on, or the requirements used in waterfall model which are fixed in the early phase. Whereas, user stories should be negotiable so that during the whole development or maintenance phase user stories could be changed according to customers' need. Additionally, user stories should also be estimatable which enables the team to better plan the development or maintenance schedule.

When the team is done with user stories gathering, estimation of user stories

²¹ <http://www.kanbanblog.com/explained/>

would be the inevitable task. The measurement unit of user stories is normally story points. A story point is referred to as an ideal day of work, which is a day without any interruption including meetings, mails, phone calls, etc. The estimation work will be done by the whole team, including both developers and customer teams. The process of estimation starts with questions and answers, with which developers know more explicitly what the customers want. When they are clear about the story content, they estimate stories by voting and re-estimating. Additionally, the estimation that has been achieved previously will be triangulated, which is to estimate a story based on its relationship with other related stories, e.g. to make sure that a 4-story-point user story is twice the effort as a user story estimated as 2 story points.

ID/Title	N137		
Description	As a customer, I want to add product in my cart, so that I can pay later.		
Estimation	4	Priority	Must Should

Figure 2.3 a user story Example with Estimation and Priority

Normally, most software projects deliver new releases every two to six months [Cohn, 2004] while some mobile or web applications release in a faster pace. And there will be different themes, which, brought up by Kent Beck, refer to a bunch of related user stories [Beck et al., 2005], concluded in each release. And when planning, the team would mostly focus on two critical questions: When to release and what to offer in this release. Concerning the first question, developers and customer may settle a range of dates instead of a specific time point over discussion. However, when there is certain product presentation, key release, etc. the release date could be fixed as well. Compared to the release date issue, the second question concerning

which stories would be included for the release is more complex. And the first problem to deal with will therefore be the prioritization of user stories. The priority could be set using just high, medium and low, or a four-level priority category including Must have, Should have, Could have and Won't have (this time), which is referred to as MoSCoW [Clegg et al., 1994].

After prioritization, the team so far would have a bunch of user story cards, formally looking like that of Figure 2.3. Concerning the efficiency and productivity of applying user story approach in agile software development, the usage of user stories in agile software maintenance process is possible as well. To better indicate and represent the variety of user requirements and maintenance requests in this study, usage of user stories is the method to utilize.

3. Agile Process in Mobile Application Maintenance

To cope with the fierce competition and rapidly changing market, constantly providing high quality application products will be the major everlasting focus for mobile software companies throughout the world. To minimize the defects and challenges of traditional software maintenance practices, the combination of agile methods and software maintenance is a promising solution. In this chapter, a theoretical agile process for mobile application maintenance is presented. In addition, some important terminologies and relevant tools are specified in details to make the description of maintenance models in the later chapters more explicit.

3.1. User Story Types and Relevant Concepts

Based on the Kano model [Kano et al., 1984], features according to user expectation are categorized into five categories, including Attractive (Exciter), One-dimensional (Linear), Must-be (Mandatory), Indifferent and Reverse. Every feature is supposed to be labeled to one of the categories above. However, indifferent and reverse features refer to those features that users don't care and users hate, which are supposed to be ignored. Thus to specify different feature stories, in this study feature stories are of three types, including mandatory feature stories, linear feature stories and exciter feature stories. The definitions are provided as follows.

- Exciter. The user does not realize he loves this feature until he sees it.
- Linear. The user considers the more of this kind of features the better.
- Mandatory. It is commonsense to contain this feature, but the user will not be satisfied without this feature.

Moreover, based on the different maintenance types which have been discussed in the previous chapter, there are three types of maintenance stories: corrective maintenance stories, perfective maintenance stories and adaptive maintenance stories. The definitions are the same to the ones presented in the previous chapter. However, as constant refactoring is considered as a routine for agile development, and also a

constant task throughout the maintenance process, it will not be considered specifically in this study.

To classify the categories, according to the method presented by Mike Cohn [Cohn, 2010], a simple user survey must be delivered including one functional question and one dysfunctional question, that is, what you think if we have this feature and what you think if we do not have it. Both questions will include 5 options, including Like, Expected, Neutral, Ok, and Dislike. And the category of certain feature concerned in the survey will be decided based on the answers of users according to the table below. And based on common statistic methodologies, it is always more accurate to involve more participants in the survey, when the option that most participants choose is the one we want to have. Furthermore, to categorize maintenance stories, the simplest method is to compare to the definition. Corrective maintenance stories refer to bugs or system crashes, while perfective maintenance stories refer to the perfection of the system. Additionally, adaptive maintenance stories cover the requirements of adaption updates concerning new hardware or system environment. Based on the definitions, maintenance stories are quite easy to categorize.

There are a number of important concepts that require to be explicated. As this thesis is concerning agile process methods, the terminologies are set according to those of Scrum. Therefore, the term ‘backlog’ is used standing for the list of ordered requirements in the formation of user stories, which is the same as that of Scrum. In order to differentiate the user stories in this study and the ones in original agile software development, there are two major types of user stories that are used here, that is, feature stories and maintenance stories. Literally, feature stories refer to the user stories which are used to indicate the requirements for new feature implementation from users. Accordingly, maintenance stories are referring to the ones that represent the requests for maintenance, perfection or enhancement for the application. The frame of feature story and maintenance story will be the same as the figure presented above. To emphasize different types of user stories and to make a

clear vision, the backlog is also divided into feature backlog and maintenance backlog, which respectively refer to backlogs with feature stories and maintenance stories. Additionally, the term ‘maintenance sprint’ is used as a basic unit of maintenance in this study according to the term used in Scrum as well.

3.2. Process Overview

The explicit methodology and framework of applying agile method principles in software maintenance practices is still under discussion. However, many articles from various sources indicated that the possibility and benefits of applying agile in software maintenance is without doubt. In [Rico, 2008], the author presented a list of the benefits of applying XP into software maintenance, including reducing code complexity and size, providing a proactive approach to problem solving, fully automating the build and test process, and so on. The benefits of applying XP in the maintenance phase will reduce the drawbacks of applying traditional software maintenance. Additionally, studies have also been conducted concerning the influence of practices in applying agile process in software maintenance and evolution based on interviews and observation [Svensson et al., 2005] presenting a quite positive result. Furthermore, in [Mello, 2012], a more specific set of guidelines was presented according to the application of agile methods in software maintenance.

Considering the motivation and benefits of applying agile methods in mobile application maintenance, and comparing with one of the agile development methods, Scrum, a concise model of agile process for mobile application maintenance is presented. The first issue to consider is the actual practice process of a mobile application maintenance process. At the end of the final release of the original version of the product, there will be feature stories left uncovered from the development phase and new maintenance stories emerged from the user reviews and team reviews as well. The actual core of maintenance is to manage and organize the backlog of both feature stories and maintenance stories. As a result of the well estimated and prioritized backlog of user stories, the team would plan the following

maintenance sprints based on certain guidelines, which would lead the whole project into a rhythm of continuous updates with the most desirable features and maintenance. The brief process is presented in Figure 3.1.

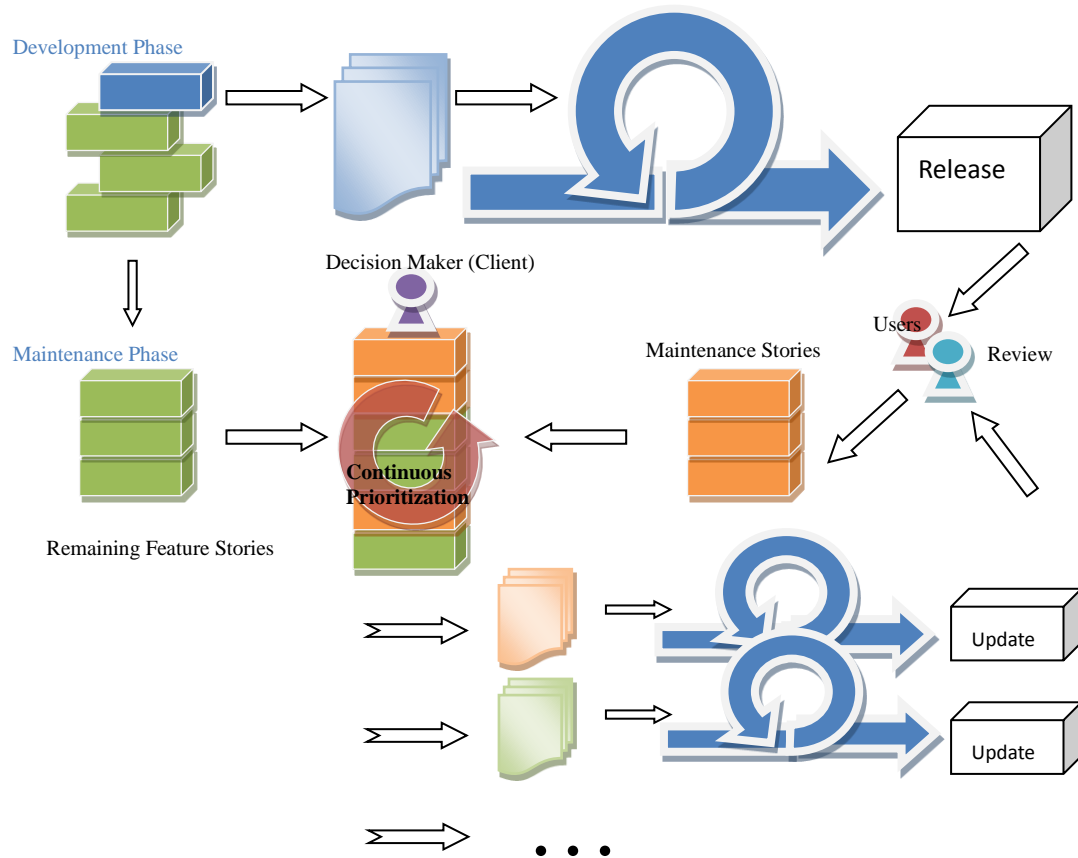


Figure 3.1 Agile Maintenance Process (using Scrum as example)

To simplify the maintenance process into certain amount of explicit steps of activities, which is also the main focus of this study, managing and organizing feature stories and maintenance stories, is the key. Therefore, in Figure 3.2, a symbolized agile process for mobile application maintenance is presented.

Different from the traditional software maintenance process, the agile mobile maintenance process includes four key steps. The first step is to analyze the situation and basic information of the project. Within the analysis, some important data will be fetched for future decision making. To follow up, based on the data analyzed previously, a suitable maintenance model will be selected to conduct the future maintenance. And based on the maintenance model and relevant guidelines, the team

as well as the decision maker will continuously work on the prioritization of the backlog and plan for the future implementation, testing and delivery. The maintenance models, when adopted, will provide guidelines for how to conduct the maintenance performance for the target application. Moreover, maintenance models are supposed to be a decent methodology that enables project team to conceive a more explicit, efficient and profitable way of performing maintenance, because even though agile methods are aiming to avoid complex frameworks and disciplines, certain instructive guidelines still improve the efficiency and profit of the whole project to a large extent. The main focus of this study is the two important steps in this process, maintenance analysis and maintenance model selection, which would be discussed in details in the following chapters.

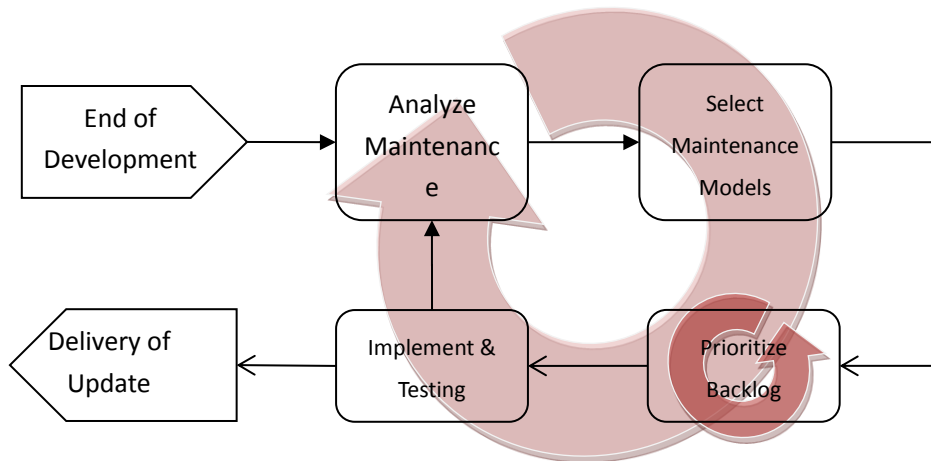


Figure 3.2 Agile Mobile Application Maintenance Process

3.3. Maintenance Board as a Tool

As in agile development, in agile maintenance phase a white board is also important for the whole crew. In the board all core information will be clearly displayed in the way that every single one member of the whole crew knows all the information he/she needs to know to cover his part of job.

An agile maintenance board could be simple but is obliged to contain the core information of the whole maintenance phase. Figure 3.3 contains one simple example

of how a basic maintenance board should look like. On the left side of the board, all user stories are placed randomly. When user stories are categorized, they are placed in either feature backlog or maintenance backlog in prioritized order. Subsequently, when the team accomplish maintenance analysis and maintenance model selection, target user stories for the next update will be selected into 'to do' column. Furthermore 'progressing' column, 'verified' column and 'done' column contain the processing user stories in respective status. Accordingly, the team could also set a 'work in progress' value like that of Kanban [Kniberg et al., 2009] and also name tags on specific user stories to indicate the responder. Those activities are optional.

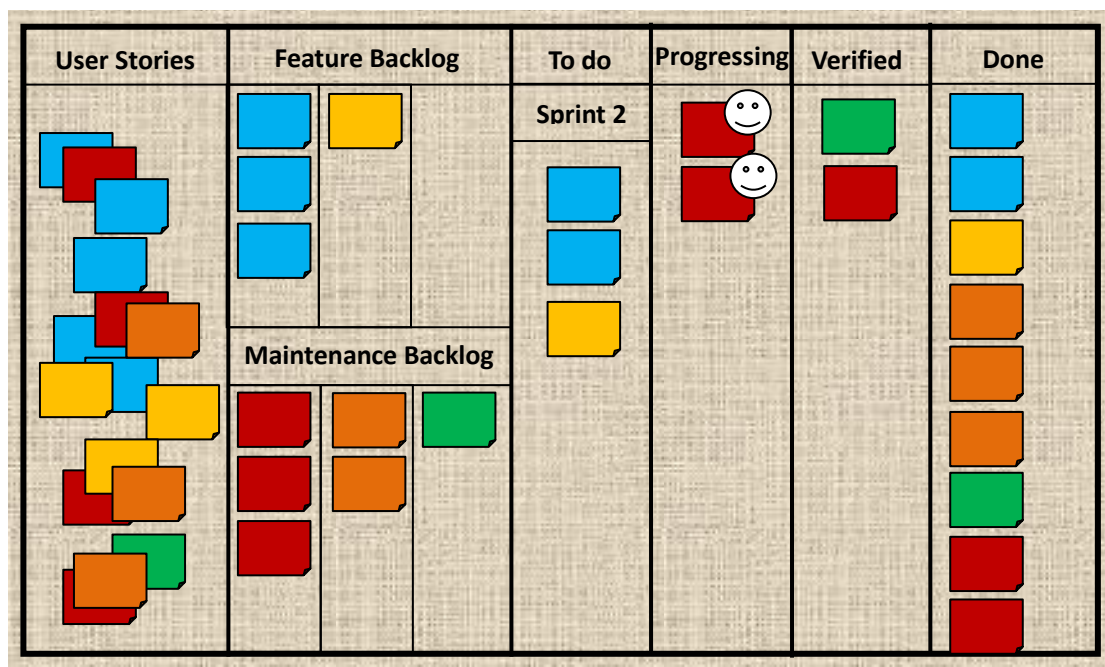


Figure 3.3 Concise Agile Maintenance Board Display

There are other important parts of the board supposed to be added in maintenance phase, including the burn-down chart, which is still quite important for agile maintenance process as well. The burn-down chart used in maintenance phase is the same as that of development phase.

4. Maintenance Analysis

Based on the findings and discussion of the previous chapter, the brief process of adopting agile mobile application maintenance is explicitly presented in Figure 3.2. To perform efficient maintenance practices, the two key steps of the process are Maintenance Analysis and Maintenance Model Selection. In maintenance analysis phase, an amount of data from various aspects will be collected and analyzed in a quantified or non-quantified way. The result of maintenance analysis will include a list of priority-labeled feature stories and maintenance stories, which are also categorized in to different types. Subsequently, in the next important step, a suitable maintenance model will be selected to execute management of product backlog and planning of the following maintenance sprint. In this very chapter, relevant details concerning the activities in maintenance analysis phase will be discussed.

4.1. Stakeholders' Focus

To succeed in manufacturing a profitable and satisfactory mobile application, the project team is obliged to be familiar with the key stakeholders of the project. And the success of the product could be defined as the maximized satisfaction from all key stakeholders. To realize the goal it is important to better dig the intrinsic value that all stakeholders are pursuing. Additionally, it is also important to better understand the relation to these key stakeholders.

The three key stakeholders in mobile application projects are developer team, client and users. Developers are the team of people who actually perform the manufacturing work while the client is equally considered as the decision maker who plays a key role in prioritizing user stories by making decisions, which sometimes could also be considered as the representative of the company owning the project. And users are the ones who are using the mobile application.

Concerning the obligations and the gains of each stakeholder, there are typical aspects of the product they are focusing respectively. And the relation of client, user

and developers and their general focus could be briefly described as in Figure 4.1.

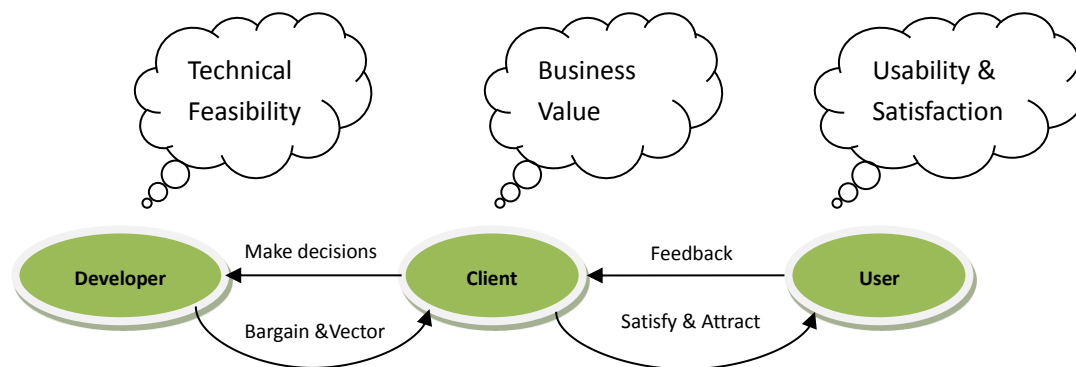


Figure 4.1 Relation of key stakeholders in agile maintenance

The figure presented above briefly shows that in agile development and maintenance the client is in a position of a crucial gear for the whole mechanism. Considering a Scrum project, the client will be the product owner, who is the representative of the client working with the team and also the connection between the team and the users. Alternatively in other projects this stakeholder might be called product manager. Obviously, the core of client's concern is the business value [Barua et al., 1995] that the product will provide. When there is somehow massive business value to realize, despite the potential risks, client would still dive for it. On the other hand, the users could not care less about the business value and how the team develops the product. They are only keen on whether the application fulfills their needs and whether they enjoy using it. Moreover, to realize the business value of the client, one of the key factors is to satisfy and attract users. Thus, whatever needs users propose, the client would still love to consider and to fulfill it. Somehow, when unable to some cross-the-line kind of requirements, the client would like to provide surprising other features to even the satisfaction, which would not surpass the technique and time limitation of developers. All the concerns of all stakeholders will be discussed and be maximally satisfied in the prioritization phase.

Concerning the three key stakeholders in one project as well as their individual focus, the success of the project will depend on satisfying the individual focuses equally. And to satisfy each stakeholder respectively, there are three individual

factors that would have key impacts on each major focus from the stakeholders, that is, User Expectation, Business Value and Risk. Thus, to prioritize feature user stories, the team and the customers are obliged to consider the weight of at least these three factors. And in order to better compare the priority of feature stories and maintenance stories, the three factors are supposed to be quantified.

4.2. User Expectation

User expectation, also considered as user satisfaction, is one of the key factors to consider when the product owner or customers decide which feature to implement first. Based on the introduction of previous chapter, all useful user stories in maintenance phase will be labeled into one of these six categories, exciter feature stories, linear feature stories, mandatory feature stories, corrective maintenance stories, perfective maintenance stories, and adaptive maintenance stories. Based on the Kano model [Kano et al., 1984] the relation of user expectation and different features included is presented in Figure 4.2 below.

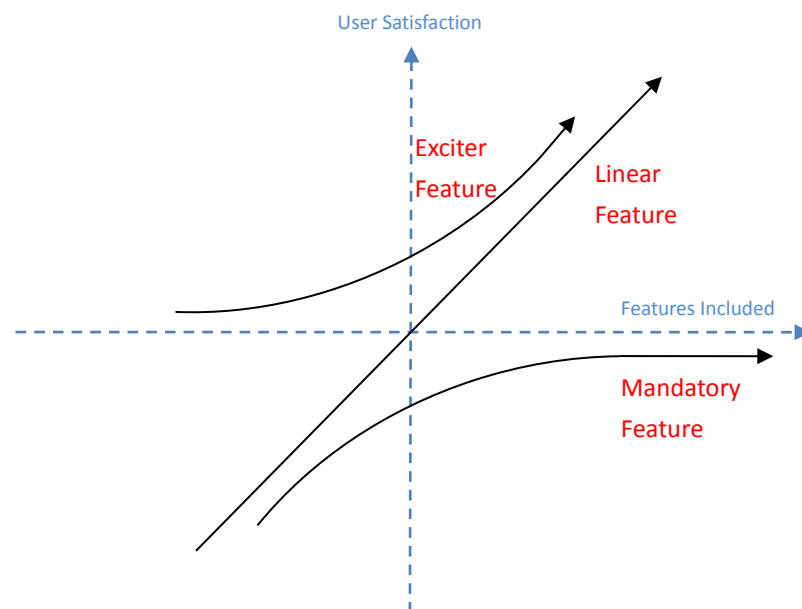


Figure 4.2 Relation of User satisfaction and Features included

The figure shows that when having more linear features implemented, users will

be increasingly satisfied. However, users will not endure lacking of mandatory features as the satisfaction drops extremely fast when mandatory features are missing. In contrast, users feel normal without exciter features but will be extremely happy to have them. On the other hand, the figure does not contain the relation between maintenance and use expectation. Adaptive maintenance is normally considered same as a mandatory feature as users will be frustrated to see an application crashing after upgrading the system. However, the situation with corrective maintenance depends mostly on how users endure them. In developers' opinion, all corrective maintenance is equally critical. Furthermore, perfective maintenance is similar to an exciter feature as users are mostly able to endure without perfection.

User review is one of the most important resources of feedbacks from end users. Some software applications on MS Windows provide a survey when users uninstall them. However, it is quite a convenient and efficient way to fetch user reviews because it is an actual survey. On the other hand, users would possibly feel annoyed when being forced to open a browser with the long boring survey. But of course it is always a good feature to provide users the option of sending reports constantly; otherwise users will never report anything, even when they are frustrated with bugs deleting the application directly. But for current mobile applications on the platform of IOS and Android, users are enabled to send reviews much more easily. By replying to users' comments, the project team would be likely to dig deeper in the various reasons for users commenting negatively and granting fewer than five stars. And some comments would possibly be organized into at least one maintenance story.

Normally user reviews will not be available during the beginning period of time right after the release of the application, as there will be limited users downloading and allocating enough time using it. Thus, it is not possible for users to respond with valuable reviews immediately. However, there will be reviews fetched from the users on site, who are already involved within development phase. They have already been involved in the acceptance testing and most likely throughout the development phase. Their reviews will be more precise and constructive.

The collection of maintenance stories is based mainly on user feedbacks. Thus, how to extract maintenance stories from user feedbacks and evaluate the importance of them is also quite important.

Firstly, user feedbacks are not always useful. For example, there are most likely a great number of feedbacks shown in Figure 4.3. What we extract from the feedbacks is generally compliments, which is nice for sure and also approval of your excellent previous development job. However, those feedbacks do not provide any constructive or innovative ideas to at least inform the team in which specific part of the application we should dig in and start fixing. Thus, we consider the similar user feedbacks ignorable. And ignorable feedbacks are simply quite acknowledgeable with similar ingredients, such as, five-star rating, short comments, etc.



Figure 4.3 Ignorable Feedbacks (screenshot from iPad)

And accordingly, some user feedbacks are not ignorable but valuable in unveiling some bugs or issues concerning the performance of the application.

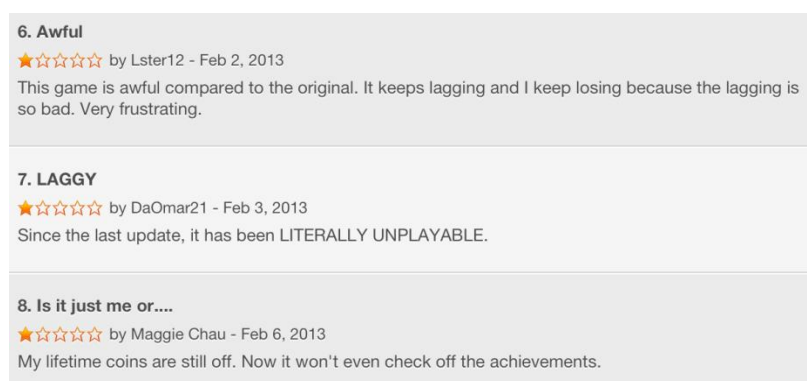


Figure 4.4 Valuable Feedbacks (screenshot from iPad)

The feedbacks presented above in Figure 4.4 are quite critical in a way, but on the other hand unveiling a typical issue of lagging of the application. And on the

basis of those feedbacks, a maintenance story '*As a user, I find the game is lagging which severely influence the performance of gameplay.*' will be generated accordingly. And to justify the severity of the maintenance story, the team should accumulate all the user feedbacks concerning the same story and simply add up numbers of the missing stars. For an instance, concerning the 3 stories in Figure 4.6, the severity of the maintenance story of lagging is counted as $4 \times 3 = 12$. And the severity of the maintenance story based on user feedbacks could also be considered as the user expectation factor on maintenance stories.

4.3. Business Value

The balance of satisfying users and obtaining business value is also crucial as the main aim of software companies is always making profit. Thus, the business value is also a key factor to influence the priority of user stories. Business value is considered somehow to be one of the most important factors that product owners and customers are pursuing during the whole project. All the features should be providing business value or at least contributing to it. And one of the simplest ways of prioritizing feature backlog is to list all user stories by business value with the one providing the most business value on the top. However, this is only for those companies or project teams which are at starting level without experience and lacking of organization. And there are some financial value should be calculated when these are values representing the business value in various aspects, including Sum of Cost, Benefits, Return on Investment (ROI), Net Present Value (NPV), Break-even Point (BP), Payback Period (PP), and so on. The calculations on these business values are determined by formulas based on certain data. And the calculation work will be done by the product owner company and delivered to the project team briefly by the decision maker.

Based on the values mentioned above, the decision makers are supposed to obtain a clear vision of how the business of this application would be realized. And there are possibly other business values, which would be possibly influenced by

specific factors. Thus, estimating explicit business value is extremely complicated. However, the details concerning the calculation of those are not compulsory for the team to know. What the team should know is the relation of different user stories in the list of business value. Thus what the decision-makers should provide is at least the list of user stories listed based on the business value from highest to lowest, which informs the team which features are the customers' favorites when the features are equally wanted by the users and of equal risk. For example, assuming that the business value one of the user stories creates is 100 while another 25, the first user story will bring approximately 4 times the value of the second one.

Additionally, the relevant business value of maintenance stories is based mainly on the feature stories that are related to the maintenance stories. As maintenance stories are created, decision makers would rate the maintenance stories with a business value rating, which could possibly be a Fibonacci Array, or from 1 to 10 rating, or 1 to 5, etc. It depends on how the company considers the importance of distinguishing stories of different business value. Or the company could also create a business rating list containing all maintenance stories and feature stories based on all the calculations above or even more details.

4.4. Risk

Taking the factors concerning various stakeholders into consideration, the client or product owner, who is about to make decisions, is obliged to think about the risks as well, as risk management [Crockford, 1986] is an integral part of software development when agile development method should contain risk management as well. When dealing with risks within an agile development project, there are various potential challenges and limitations, including resource limitations, implicit correspondent risk owner, direct relation to the backlog, acceptance of residual risks, and so on [Ylimannela, 2012]. As whenever a risk is detected, it is important to document it or attach a quick solution to it for future efficiency. The brief process of identifying and recording risks is described in [Ylimannela, 2012].

According to the analysis process we can briefly know how to deal with emerging risks. For low severity risks, they should be identified and responded quickly by brainstorming. And the possible solution will be noted down next to the feature stories in order to address the risk within the process of implementing this feature. However, when the risk is severe enough, more time is allowed to allocate on working on the solutions. And there is a limitation of time consuming, when surpass it the project should move on leaving the risk into maintenance backlog considered as preventive maintenance of high priority.

The most significant factors determining the severity of risks are impacts and probability, that is, the risks that would result in more serious consequences and are more likely to occur will be considered more serious, which is obvious. The formula of calculating severity of each risk item is

$$\text{Severity} = \text{Impact} \times \text{Probability}$$

During the process of analyzing potential risks of the project, a risk list will be created including all the potential risks within the process of maintenance phase marked with impact rating and probability rating, and also relevant solutions. The elements on the list should be adaptable, as the impacts and probability of risks are shifting when maintenance proceeds while there are possibly new risks emerging as well.

Risk ID	Description	Impact	Probability	Solution
R1	Personnel shortfalls	4	3	
R2	High Coupling with multiple components	3	3	
R3	...			

Table 4.1 Risk list sample

Table 4.1 presents an example of risk lists with impact and probability rating. The risk items in the list are prioritized in the order of severity. The impact and probability rating could be from 1 to 5, or 1 to 10.

When an accomplish risk list is under proper updating pace, and a business value list as well, based on the maintenance stories extracted from user feedbacks, the team could calculate the rough priority by applying the formula as below.

$$Priority = \frac{User\ Expectation\ Rating \times Business\ Value\ Rating}{Risk\ Rating}$$

The core principle is that the priority of any user story is directly proportional to user expectation rating and business value rating, and is inversely proportional to risk rating. The priority of maintenance stories will be considered as a key factor for the team and decision makers in choosing the right ones for the coming maintenance sprint. However, concerning the maintenance model that the decision makers choose, there will be slight changes in selections. In the following chapter, more details will be presented concerning the selection of suitable maintenance features based on priorities when using different maintenance models.

4.5. Other Relevant Information

Besides the major factors presented above, there is other important information of the project and the target application product that the project team should be familiar with. That information will influence the process of execution of the maintenance in many ways. The attributes of the project are important factors that the team should be familiar with during the whole development and maintenance period. Mostly, the team should and is obliged to get to know their employer at the very beginning of the project. However, somehow the knowledge would possibly float on the surface. When these facts are not understood by the team, there will be decisions from the client based on these attributes, which possibly result in the misunderstanding and argument within the group.

- Team Working Rate

Velocity is a key factor that should be tracked within the whole development and maintenance when defining the efficiency of the maintenance team. When the project enters maintenance phase, the velocity of the team should have been added up into statistics, which was shifted from iteration to iteration but converged to be constant

so far. The velocity standard will be the according efficiency of the team, which is critical for the latter planning for maintenance. In addition, there is somehow a slight chance that the velocity in maintenance phase changes sharply according to various factors, including different types of maintenance cases, percentage of new feature development of maintenance iterations, etc. Velocity should always be tracked and adjusted from time to time.

- Business Models

Based on the information presented in previous chapters, there are commonly 7 different types of business models in mobile application business, which could also be considered as 7 different ways of earning profits, including application purchase, in-app purchase, freemium, advertising, free, subscription and donation [Vannieuwenborg et al., 2012] [Arrayent, 2012]. Choosing a different business model would most likely divert the market direction of mobile application as well as the maintenance work. Normally one application would only adopt one business model to provide profits as users would probably not tolerate more than one way of asking for payment and would consider the company an impulsive money-absorber.

- Client participation

This information answers the question who is the actual decision maker in the prioritization process. According to the size of the company or organization from which the client is, the level of client participation shifts tremendously. Big companies with over 200 employees seemingly tend to make decisions themselves only considering developers' opinion as advices and tips. On the other hand, clients from small companies with no more than 52 employees are more willing to grant authorities to the developer teams on making decisions, especially if they are lacking relevant knowledge about the product they are demanding [Racheva et al., 2010].

- Resource Availability

Small company clients with limited resources on small projects will have to focus on the core features and the guaranteed performance of those features without considering other new exciting features which might bring business incomes as well

as intangible risks. On the other hand, big companies would to large extent love to accept other features to bring more business value.

- Value Criteria

Value criteria are concerning the evaluation of the value for clients, that is, the value of which category is mostly considered important to the clients. It is also a decision making factor, which should be also considered by the decision makers. Continuous communication will help to track the change of value criteria of the client as well.

However, the information mentioned above is only side-factors to influence the process. What is the relation between each individual side-factor and the efficiency and productivity of the project is not yet concerned in the current study. That would most likely be one of the future works to be considered. Additionally, the information concerning project attributes is not completely transparent to the whole team as it is not relevant. Regarding the decision makers, they should keep the business-related information in a “black box”, only revealing limited information, such as business model and resource limitation and so on. The maintenance team will focus on the core business value realization.

5. Mobile Application Maintenance Models

When the analysis of maintenance has been done according to the instruction in previous chapter and the existing feature stories are well prepared, the next important step is to plan for the maintenance sprint by selecting user stories from both feature backlogs and maintenance backlogs, which are both well prioritized and managed. How to select the user stories that will satisfy and excite most users and provide most benefit to the company is the key concern.

Based on the observation to the update history of the chosen mobile applications, there are some interesting facts deserving to be noticed. About 33% of the applications update at a random pace while the rest update regularly. Furthermore, the applications that update at a random pace deliver nearly no new features but bug fixes in most of the updates when the regularly updating applications provide new features, bug fixes and performance improvement constantly. For example, Foursquare²² updates nearly once in two weeks with constant new feature deliveries and enhancement as well. On the contrary, Renaissance of Civilizations²³ updates only four times in a year. Among the regularly updating applications, some game applications provide different themes and additional tasks in each update. Via further analysis of the cases, certain regular patterns were conceived according to different applications and were summarized into maintenance models.

In this chapter, three different mobile application maintenance models are proposed based on the study and analysis of the chosen cases. The maintenance models are categorized in terms of the update frequency, update content and the factors that influence the decision maker to choose the update content. According to the previous discussion on the process of agile mobile application maintenance, the selection of mobile application maintenance models is an important step as the maintenance model will to large extent determine the update pace, update content, and the efficiency and productivity of the project.

²² <https://foursquare.com/>

²³ <https://itunes.apple.com/cn/app/wen-ming-fu-xing/id467940664?mt=8>

On the other hand, depending on the various types of applications as well as different goals and perspectives of the organizations, the corresponding maintenance models of it are quite diverse. It is not justified to verdict that certain model is worse than another. However, it would be more efficient and organized to perform a model in high level disciplines and regulations. Depending on typical circumstances, we are presenting three respective maintenance model models as well as one or more example application adopting them.

5.1. Emergency-Oriented Maintenance Model

The main aim of Emergency-Oriented Maintenance Model is to deal with various types of emergent issues within maintenance phase including bugs, system crashes, environment adaption, etc. as soon as possible. One of the advantages of this model is the maximized stability of applications running smoothly in all possible environments while the disadvantages is lack of planning, fixed schedule, new features and sometimes motivation.

The process of Emergency-Oriented Maintenance Model is to a large extent random and spontaneous. The whole maintenance process is more likely the extension of the debugging of previous development phase. Together with continuous testing, debugging and refactoring, as well as incoming user reviews, there are more potential bugs emerging. Thus, fixing bugs is of the highest priority without doubt when decision makers and developers would set the actual priority and effort of each maintenance story via discussion and voting. Subsequently, when the maintenance starts, maintainers are supposed to cover at least critical corrective maintenance stories first, including some perfective maintenance stories if it is possible. Additionally, whenever adaptive maintenance occurs, it should be promoted to the level of corrective maintenance stories. Correspondingly, the release date is depending on the accomplished amount of bugs fixed and the degree of urgency from users.

A concisely described process of Emergency-Oriented Maintenance Model is

described as Figure 5.1.

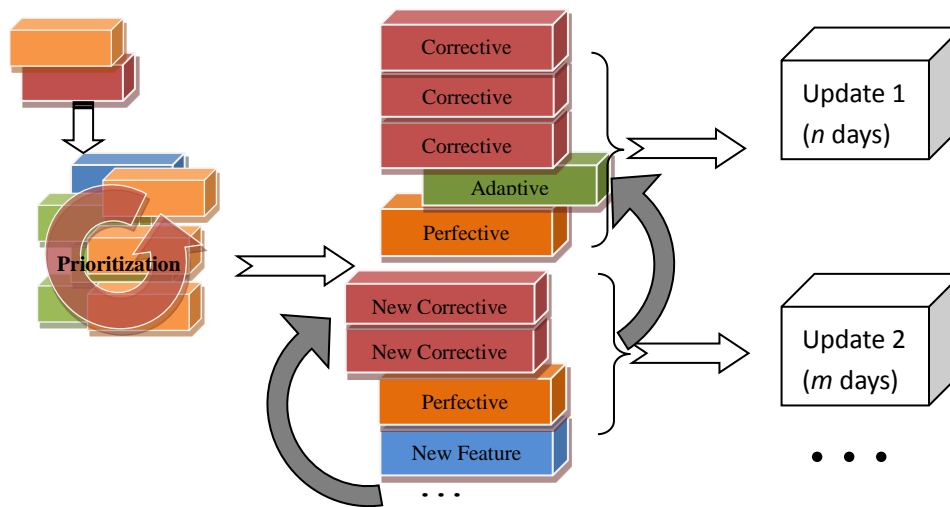


Figure 5.1 Emergency-Oriented Maintenance Model

Dark Runner²⁴ is an action video game developed by Dumadu Games²⁵, an Indian software company aiming in software solution and game development for mobile devices. This game is easy to catch up but challenging with the only mission to make the runner survive by avoiding obstacles on the ground and in the air. There are only two action controls, jump and roll. The game was first released in April 2011. There are five minor updates through the year 2011, as presented in Table 5.1

These updates did not cover any mandatory feature changes with only two exciter features added in one year. In addition, there are also limited perfective maintenance features as the graphic is two-dimensional with only black and white colors including the icon.

v1.1	01/07/2011	• Integrated iAds (Perfective);
v1.2	11/07/2011	• Integrated iAds and improved performance (Perfective)
v1.3	29/07/2011	• Bug Fixes(Corrective); • Background music changed (Perfective); • Local device scoring system implemented (Linear)
v1.4	30/09/2011	• Check leaders board (Exciter); • Optimized gameplay (Perfective); • Game Feed (Exciter);
v1.5	22/02/2012	• Retina support (Perfective); • Improved Performance (Perfective);

²⁴ <https://itunes.apple.com/us/app/dark-runner/id430822832?mt=8>

²⁵ <http://dumadu.com/>

		• Tested on IOS5 (Adaptive)
v1.6	11/03/2012	• Bug fixed and performance improved;
v1.7	26/04/2012	• Bug fixed;

Table 5.1 Update History of Line Runner

In 2012, there were another two updates in March and April containing only slight perfective and corrective maintenance. The following update came eight months later including only perfective as well. As the game is limited by the game style adopted and graphics, there is limited space for perfective updates and linear features (e.g. new levels, etc.). Thus, Emergency-Oriented Maintenance Model is quite applicable for this application when the maintenance team requires not too much time and effort spent in monitoring and updating. The fact is that only perfective maintenance is covered in most of the updates, as well as some bug fixing, which is based on the feedbacks of users. Thus, the maintenance process depends on whenever the team and client feel that certain emergency situation occurs and it is time to perform serious maintenance.

Another case of adopting Emergency-Oriented Maintenance Model is another video board game on IOS platform, named JiShang Splendid Mahjong²⁶ (in Chinese ‘极上豪华麻将’), and developed by Auer Media & Entertainment Corp²⁷. There was only one adaptive update within the year 2012 to fix compatibility with IOS6 and some relevant bugs in October. And it took place in the blossom season for IOS6 updates.

By selecting Emergency-Oriented Maintenance Model, project teams will save a considerable amount of time and effort, which will be spent on more important other applications or the invention of new applications. Additionally, the minor emergency-oriented updates to a large extent maintain the stability of the application product, which would minimize the possibility of risks of triggering new bugs. However, on the other hand, the whole process of maintenance using Emergency-Oriented Maintenance Model would lack certain disciplines. And there will be very limited or no new features to excite and attract users, which could

²⁶ <https://itunes.apple.com/cn/app/ji-shang-hao-hua-ma-jiang/id422808176?mt=8>

²⁷ <http://www.auer.com.tw/index.jsp?lang=en-us>

possibly reduce the motivation of the project team as well.

5.2. Event-Oriented Maintenance Model

In Event-Oriented Maintenance Model, the first question is what kind of features are supposed to be considered as event-related or seasonal features. There is no explicit definition so far in literatures as it is still a new term that is brought up in this thesis. In this study, the concise and ambiguous definition of this term could be all features and relevant adjustment that are only developed during a specific period of time according to the events or season around that moment. The event-related and seasonal features are meant to provide users different themes for the game environment to avoid aesthetic fatigue. Additional rewards and tasks are also stimulators. Basically there are several typical aspects which are most likely to be considered in an event-related or seasonal update, including graphic theme, game items, characters, tasks, rewards, among which the most important aspect is the graphic theme. Dramatic seasonal theme could be seen switching from the game Subway Surfers²⁸ in Figure 5.2.



Figure 5.2 Theme changes of Subway Surfers (screenshots from iPad)

Compared with Emergency-Oriented Maintenance Model, Event-Oriented Maintenance Model is more organized and containing disciplines and regulations. The main aim of this maintenance model is to provide seasonal updates or/and special updates containing exciter features or linear features for certain major events

²⁸ <https://itunes.apple.com/fi/app/subway-surfers/id512939461?mt=8>

or festivals, e.g. Christmas, New Year, Halloween, etc. However, Event-Oriented Maintenance Model should be combined with Emergency-Oriented Maintenance Model as there are always bugs or unsatisfactory performance at unexpected time requiring to be dealt with. It will trigger users' limit of endurance when updates containing relevant solutions are released only on certain occasions. Additionally, this model is more applicable for games than other applications.

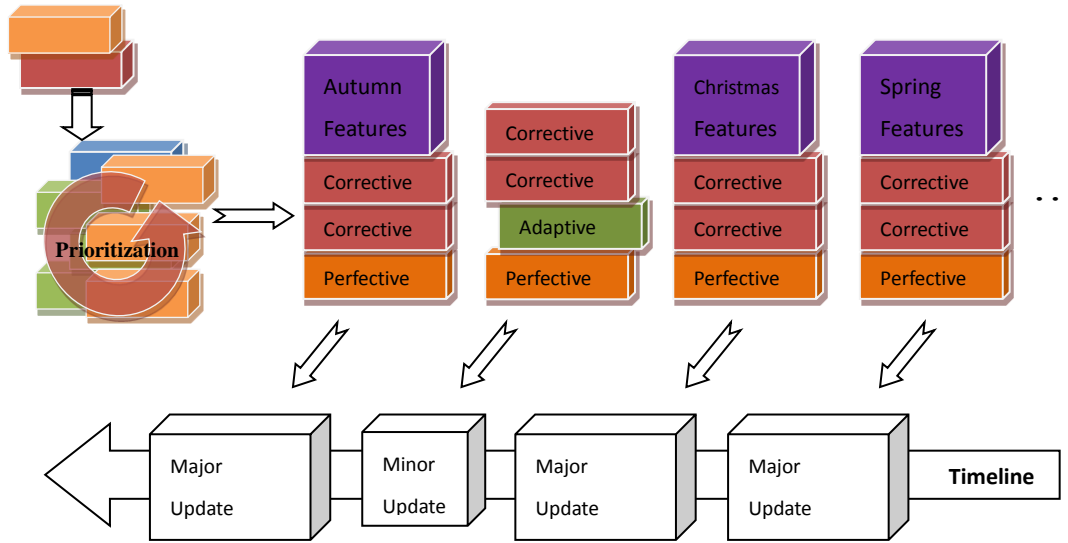


Figure 5.3 Event-Oriented Maintenance Model

The process of Event-Oriented Maintenance Model is different from Emergency-Oriented Maintenance Model based on the differences in their aiming targets. Event-Oriented Maintenance Model intends to provide distinguishing features and enhancement at particular season periods. Mostly, a season update or event update would contain many seasonal features and all corrective and perfective maintenances that are obliged to be dealt with in this season or not urgent enough to be covered in the previous minor update. Correspondingly, there would be also somehow minor updates needed when there are urgent critical bugs to be fixed. At the end of releasing a seasonal or event update, the team and the client should settle down with a deadline for the next season or event in the retrospect or planning meeting. The team will focus on the high prioritized new exciter features for the coming season or event unless there are critical bugs or adaptive maintenance of

sufficiently high priority to plan a minor update. A new bug reported would be dealt firstly when the decision maker or the team believes it is critical enough to start a minor release. Other perfective maintenance stories are supposed to be put at the end of the priority line. They will be covered when season features are ready to ship.

Subway Surfers is also an action video game presented by Kiloo²⁹, a Danish software company. The game was first released on May 24, 2012 with the main feature to avoid being caught by the police by dodging, rolling and jumping. Different versions, as well as update contents and date, are presented in Table 5.2.

V1.2.0	24/05/2012	<ul style="list-style-type: none"> • Five new characters to unlock (Linear) • New exclusive trophies to find (Linear) • Unique Game Center Achievements (Perfective) • Fixed startup bugs (Corrective) • Facebook login reward (Linear) • Various bug fixes (Corrective) • Graphic, Stability, Performance Improvement (Perfective)
V1.2.1	28/07/2012	<ul style="list-style-type: none"> • World rotation removed (Perfective) • Minor bug fixes (Corrective)
V1.3.0	15/09/2012	<ul style="list-style-type: none"> • Unlock 5 fresh and cool characters (Linear) • Complete new Missions (Linear) • Get awesome daily rewards (Linear) • Collect improved Super Mystery Boxes (Linear) • Improved startup (Perfective) • Gameplay optimizations (Perfective) • Various bug fixes (Corrective)
V1.4.0	24/10/2012	<ul style="list-style-type: none"> • ‘Spooktacular’ Halloween world (Event Feature) • Special Zombie Jake character only available for a limited time (Event Feature) • Gets you in the mood for the Halloween holidays (Event Linear Feature) • Improved startup (Perfective) • New Character Selection screen (Perfective) • New graphics and animations (Perfective) • Switch between Halloween and classic look from Settings menu (Perfective)
V1.5.0	28/11/2012	<ul style="list-style-type: none"> • Snowy Winter Wonderland (Season Feature) • Elf Tricky character only available for a limited time (Season Linear Feature) • New Boards Surf with style and discover awesome powers (Linear Feature) • iPhone 5, iPod 5 optimization (Adaptive) • Free gift for the holidays (Linear) • Restyled menu (Perfective) • Improved performance (Perfective) • New graphics and animations (Perfective)
V1.5.2	11/12/2012	<ul style="list-style-type: none"> • Critical performance fix (Corrective/Perfective)
V1.6.0	02/01/2013	<ul style="list-style-type: none"> • Join the Subway Surfers World Tour First stop: New York City (Season Feature) • Explore the NYC Subway (Season Feature)

²⁹ <http://kiloo.com/games/subway-surfers>

		<ul style="list-style-type: none"> • Get the street smart kid, Tony, for a limited time only (Season Linear Feature) • Expand your board collection with 4 new beautiful boards (Linear)
--	--	--

Table 5.2 Update History of Subway Surfers

Seen from the table above, at certain typical time, e.g. Halloween, Christmas, etc. the game was updated with corresponding themes and some relevant linear features. And the date of release is basically according to the season but not to a specific date in a month or week. In addition, between two major updates, there are occasionally certain minor updates to cover critical bug fixes and performance enhancements (e.g. V1.2.1 and V1.5.2). The release dates of minor updates are even more spontaneous. Thus, it is a classic example of Event-Oriented Maintenance Model combined with Emergency-Oriented Maintenance Model. This combined model copes with emerging bugs and flawed performance in time which minimizes the possibility of losing existing users, while the season features and event features are largely increasing user satisfaction as well.

Another example is mobile video game Clash of Clans, which was mentioned previously. The game was first released on August 2nd 2012. Within the half a year of maintenance, there are 6 major updates without minor emergency updates, that is, nearly once in a month. In each updates there are a number of new linear features included as well as certain amount of bug fixes and performance enhancements. Additionally, winter theme was adopted in Version 2.111 including special linear features. The different aspect of the typical model is that the team enables updates in a suitable pace so that bugs are supposed to be fixed in time while new linear features and exciter features are delivered constantly.

5.3. Constant Maintenance Model

Constant Maintenance Model is a regular but disciplined maintenance model that is suitable for most of the teams and projects. Compared to the other two maintenance models mentioned above, this model is more likely in a style of “agile-disciplined” manners. The whole maintenance process is constantly divided into maintenance

sprints with durations of two or three weeks. Each of the maintenance sprints will cover corrective maintenance of highest priority as much as possible and mandatory features if missing. When there are no urgent corrective maintenance stories to cover or mandatory features to add, adaptive maintenance would be the next to cope with, followed by linear features, exciter features and perfective maintenance, which are nearly of the same priority level. When concerning specific stories, the priority will be further discussed and voted by team and decision maker based on the three factors mentioned in previous section including user expectation, business value and risk.

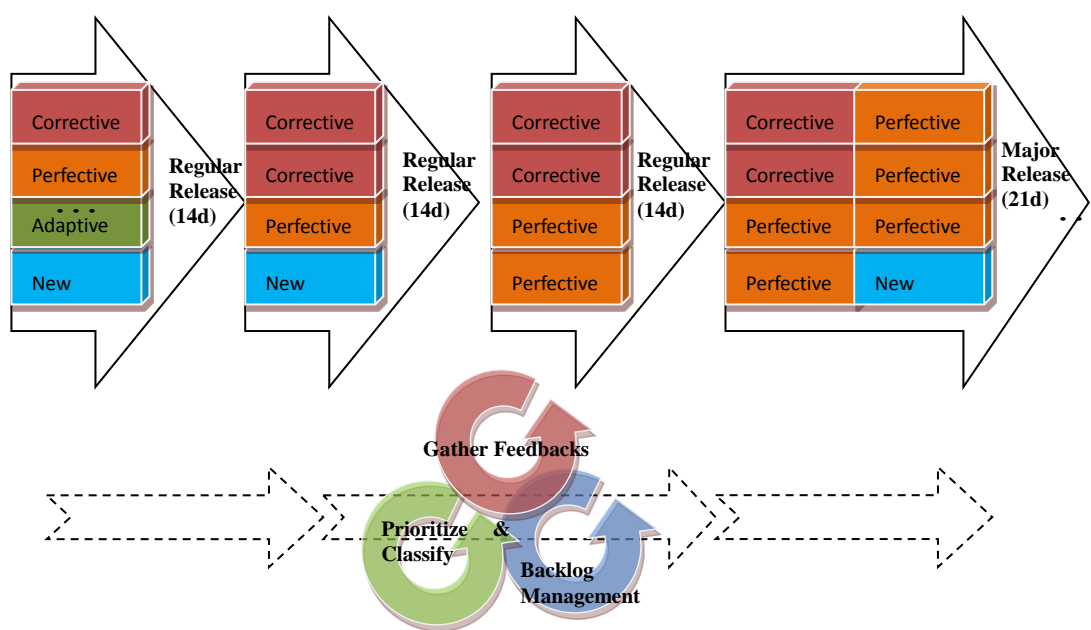


Figure 5.4 Constant Maintenance Process

Constant Maintenance Model is a regular option for most of the mobile applications on IOS. It provides constant updates within relevantly short period of time. Compared to Event-Oriented Maintenance Model, commonly applications updated with Constant Maintenance Model set each maintenance sprint as two or three weeks, even longer. The whole maintenance process seems to be more flow-like which is more agilely organized.

The brief description of the process of Constant Maintenance Model is presented in Figure 5.4 above. Basically, the process of Constant Maintenance Model is to large extent identical to the process of agile software development in the way that contains certain circulating activities in order to accomplish small tasks constantly and

achieve targets in typical phases. According to various circumstances concerning the company or the team, the length of each maintenance sprint is ranging from 10 days to three weeks, including weekends and statutory holidays. Before starting one maintenance sprint, a planning meeting would be preferred in order to discuss mainly the target of this sprint. There are certain issues obliged to be settled within this meeting, including:

- Well prioritized and classified backlog
- The target maintenance stories or new feature stories in the following sprint
- Assignment allocated

Foursquare is one of the most popular social platforms throughout the world. The core feature of this application is to share current location as well as personal tips concerning the location with friends. Users are also enabled to share instant photos of the location. Additionally, users receive badges as rewards and become majors of locations when they check in the most times in one location. The majors of certain locations, for instances restaurants, cafés, clubs, or bars, and so on, might obtain certain discount or other kinds of benefits. The first version of Foursquare on IOS was released on March 7th, 2009, and the latest version is 5.4.1 released on January 12th, 2013. The table below provides details concerning all updates since version 5.2 in the last half a year.

V5.2	• 17/07/2012	<ul style="list-style-type: none"> • A bunch of bug fixes (Corrective) • Graphic updates with surprises (Perfective/ Exciters)
V5.2.1	• 27/07/2012 8 days	<ul style="list-style-type: none"> • A bunch of bug fixes (Corrective)
V5.2.2	• 09/08/2012 9 days	<ul style="list-style-type: none"> • Bug fixes (Corrective) • Enhance performance (Perfective) • Nearby friends check-in feature is back based on user feedbacks (Mandatory)
V5.2.3	• 24/08/2012 • 11 days	<ul style="list-style-type: none"> • Bug fixes (Corrective) • Enhance Performance (Perfective)
V5.2.4	• 07/09/2012 10 days	<ul style="list-style-type: none"> • Make easier to share check-ins in Facebook and Twitter by moving the options up to the top of the screen (Perfective) • A bunch of design in UI and under-the-hood improvement (Perfective)
V5.3.1	• 01/10/2012 16 days	<ul style="list-style-type: none"> • Map Perfection (Perfective) • Explore tab update with a simpler design (Perfective) • New search categories to make searching places easier (Linear/Exciter) • Instantly see friends and top suggestions nearby (Linear)

		<ul style="list-style-type: none"> • Search for specific places (Linear) • Search for whatever you're craving (Linear) • A bunch of design tweaks and under-the-hood improvements (Perfective)
V5.3.2	<ul style="list-style-type: none"> • 17/10/2012 12 days 	<ul style="list-style-type: none"> • Bug fixes (Corrective) • Speed up check-ins (Perfective)
V5.3.3	<ul style="list-style-type: none"> • 05/11/2012 13 days 	<ul style="list-style-type: none"> • Easier way of finding best places by adding score next to the name of the place, which provide a quick sense of how much people love it (Exciter) • Better ways of observing more accurate scores (Perfective)
V5.3.4	<ul style="list-style-type: none"> • 20/11/2012 11 days 	<ul style="list-style-type: none"> • Add 'Recent Opened' search option (Exciter)
V5.3.5	<ul style="list-style-type: none"> • 04/12/2012 10 days 	<ul style="list-style-type: none"> • Palindromic version (Perfective)
V5.4	<ul style="list-style-type: none"> • 17/12/2012 9 days 	<ul style="list-style-type: none"> • New feature to help deciding where to go and what to do. When looking for a place, users are about to see important stuff up top such as ratings, address, phone numbers, etc. When checked in, users will see nearby users, tips, photos at top to obtain first impression (Exciter)
V5.4.1	<ul style="list-style-type: none"> • 12/01/2013 19 days 	<ul style="list-style-type: none"> • Easier feature to add friends (Perfective)

Table 5.3 Update History of Foursquare

Based on the table above, we can obviously observe the maintenance model adopted by the team of Foursquare. Firstly, the regular length of each maintenance sprint is around 10 working days. However, in some cases when the work in the sprint is not intense but the remaining time is insufficient to cover another feature story, the update will be released in eight or nine days. On the other hand, when a major exciter feature update is included in the sprint, the sprint would last longer up to 13 even 16 days. When that the latest sprint period covered Christmas and New Year, the duration turned out to be 19 days.

Within all these updates mentioned above, most updates covered bug fixes and performance enhancement, which are the stories of high priorities. As the length of each maintenance sprint is short, whenever there was an exciter features update there would be no other maintenance covered. Most likely, the team is keeping the maintenance pace well enough in retaining the stability of the system as well as providing linear features and exciter features occasionally.

6. Guidelines for Planning the Use of Maintenance Models

After deciding which model to apply in the maintenance phase, the team and decision makers are obliged to carry out a general plan to perform maintenance in an efficient and profitable way. The core of the plan is to prioritize the whole backlogs including all feature stories and maintenance stories. Obviously the factors taken into account which would to large extent influence the priority of stories are the ones mentioned in the previous section, User expectation, Business value and Risks. According to the formula we presented in a previous chapter, that is

$$Priority = \frac{User\ Expectation\ Rating \times Business\ Value\ Rating}{Risk\ Rating} \times Custom\ Weight$$

We are able to prioritize maintenance stories as well as feature stories and also to see the contrast between either two of them. Concerning the different maintenance models applied, the weight of certain factor influencing the priority of certain types of story would shift dramatically. For instance, when selecting Emergency-Oriented Maintenance Model, a decision maker would rather increase the weight of risk rating in case the maintenance of high risk will be postponed. On the other hand for certain application based on Event-Oriented Maintenance Model, the user expectation of event-related features will be increased accordingly. And factors weigh also differently concerning feature stories and maintenance stories of different types. However, in this study quantified details and formulas will not be discussed concerning how the custom weight should be set according to various circumstances. In this chapter, we propose a set of general guidelines covering how to set priorities for various user stories in different maintenance models.

6.1. Emergency-Oriented Maintenance Model

When choosing this maintenance model, the company has decided that the application is currently remaining in one of or a combination of couple of situations

mentioned in the previous section concerning the suitable applications adopting this maintenance model. The decision makers are supposed to be aware of the situation that users require no more new features from this application. The core mandatory features have already been considered as being at a quite satisfactory standard so that no more updates are required for this part. Therefore, during the process of adopting Emergency-Oriented Maintenance Model, the team would only focus on different types of maintenance stories, including corrective maintenance stories, adaptive maintenance stories and perfective maintenance stories.

There are some general guidelines concerning in what circumstances Emergency-Oriented Maintenance Model is suitable for the target project.

- The application contains sufficient core features to provide satisfactory value;
- The application contains bugs that would be so ignorable that majority of the users would not notice till the update;
- The application requires no extra linear features to keep satisfying existing users;
- The client has limited patience on this application because there are a number of other applications requiring urgent maintenance;
- The funds are limited;
- The application has accomplished its mission by earning enough profit, attracting media coverage or promoting company's reputation;
- Starting team with little experience in managing the process;

The core of Emergency-Oriented Maintenance Model is to deal with urgent bugs and perfective maintenance requirements as soon as possible in order to ease the dissatisfaction of users. Based on the statistics of user feedbacks from IOS platform, most of the lowest ratings concern corrective bugs. Thus, in a priority list of an Emergency-Oriented project, corrective maintenance stories should always be considered to be put in front. Perfective maintenance should be subsequently allocated downward. Adaptive maintenance stories normally occur only when there is a major update in a mobile system, e.g. IOS6 recently. And when there are

requirements in adapting application to new system platform, adaptive maintenance stories should be prioritized as important as corrective maintenance stories. To prioritize maintenance stories from the same type, the formula below will be used in the following instruction.

User expectation rating is the sum of the missing stars rated concerning the same corrective maintenance stories, which is described in the following equation.

$$User\ Expectation\ Rating = \sum (5 - Star\ Rating)$$

The business value rating will be based on the rating provided by decision makers or the business value rating list. The risk rating is the sum of the risk severity of all concerned risks, that is,

$$Risk\ Rating = \sum (Impact \times Probability)$$

When the priorities of all maintenance stories are set, the team should also estimate the effort of each maintenance stories as well. The method of estimating effort is the same as that in development phase. And time schedule will be the next task. The first step is to settle with a deliver deadline. As far as we know, the applications adopting Emergency-Oriented Maintenance Model normally do not have a fixed deliver deadline. However, it does not mean that the team is totally idle when there is no delivery pressure. Team members are most likely focusing on other applications development as well. Thus, only when the decision makers decide it is time to release an update as some bugs reported by users are urgent enough, the team is gathered to deal with this matter. Based on the team working rate and prioritized and estimated maintenance stories, decision makers and the team would quickly settle with the story selection to cover the following update. When there are not enough corrective maintenance stories to cope with, perfective maintenance stories of high priorities are brought in as substitution. And when the actual maintenance phase starts, even though there are new updates of user feedbacks which would change the priority rating of some maintenance stories, the selected stories will be continued without interruption.

To sum up, the simplest way to adopt Emergency-Oriented Maintenance Model

is to follow the guidelines below.

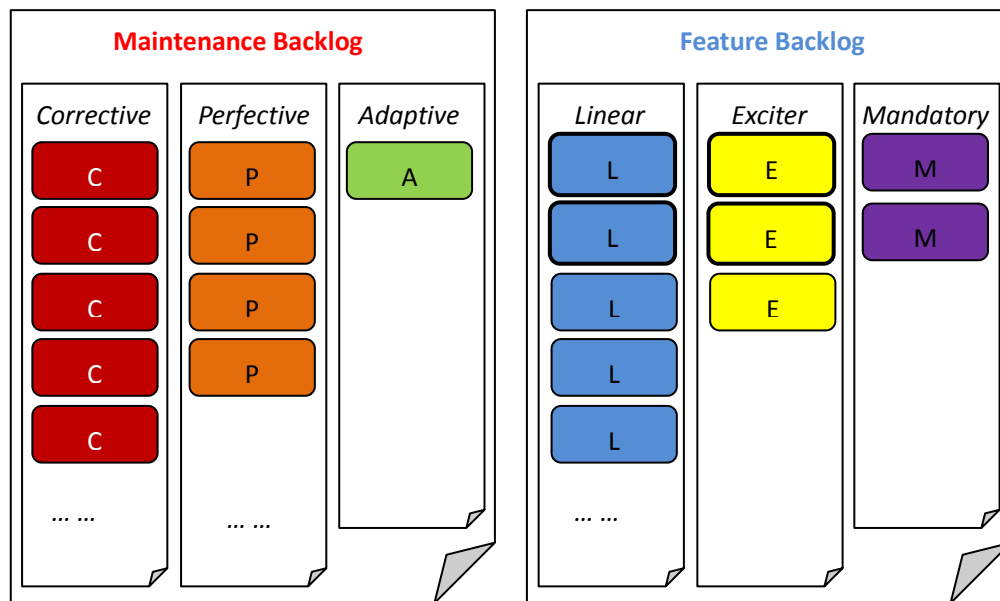
- Extract maintenance stories from user feedback, categorize and estimate them.
- No feature stories but mandatory feature stories are covered.
- Adaptive maintenance stories should be prioritized as high as corrective maintenance stories; and corrective maintenance stories should be prioritized higher than perfective ones.
- Only put perfective maintenance stories before corrective ones when the rest of the corrective maintenance stories could not fit the time-box.

6.2. Event-Oriented Maintenance Model

Compared with Emergency-Oriented Model, Event-Oriented Maintenance Model is to large extent organized and optimized. The main content of this maintenance model is to provide event-related or seasonal features and updates within a certain period of time, accompanied with certain corrective and perfective maintenance solutions. The aim is to sustain continuous interest of existing users and keep the application in a form of sustainable development with continuous profit. Event-Oriented Maintenance Model is normally adopted in game applications, which is based on the characteristics of games. The suitable applications to adopt Event-Oriented Maintenance Model include:

- Game applications including various levels with increasing difficulties;
- Game applications with themes that would vary from time to time;
- Applications with customized themes to choose;

Thus, based on the overview of the maintenance model process, there are some specific details that we should dig a bit into. The first step to start the maintenance is to extract user stories from user feedbacks, just like Emergency-Oriented Maintenance Model. Similarly, the maintenance stories will be categorized into corrective, perfective and adaptive maintenance stories. All the maintenance stories will be listed according to types in maintenance backlog.



**Thickly edged feature stories are event-related.*

Figure 6.1 Categorized User Stories

Feature stories are categorized according to the different types of features mentioned in previous section, including linear features, exciter features and mandatory features. All feature stories will be listed in a separate feature backlog, which is like the one presented in Figure 6.1 above. In each type of user story list, the stories will be estimated and prioritized. The method of estimating potential effort is the same in all user stories. To prioritize those stories, the team would still adopt the general priority formula concerning user expectation, business value and risk, mentioned in the previous section. However, to emphasize the event-oriented or seasonal updates, event-related features and perfections would be more highly weighed in feature backlog in order to keep those event-related feature stories in front. The weight of an event could vary depending on the importance of those event-related features in this maintenance sprint.

Subsequently, depending on the working rate of the team, a suitable time-box should be set at the beginning of the first maintenance sprint. Based on observation on current top mobile games on IOS, the time-box is approximately a month or two. There are also customized reasons for companies to choose a more suitable duration

for each update, including most importantly the actual event-related content of the coming update. The time-box is flexible in a way so that at least all corrective maintenance stories and event-related features are included. The exact release date will be discussed during the maintenance sprint and set finally by decision makers.

When prioritizing feature stories and maintenance stories into one sprint backlog, there are some general issues that would possibly occur and should be taken care of.

Mandatory feature stories and adaptive maintenance stories prevails

Whenever there are mandatory feature stories and adaptive maintenance stories in backlogs, they should be considered of highest priorities in any maintenance sprint.

The problem is quite common when the team is trying to achieve all event-related and seasonal updates and leaving limited available time for corrective and perfective maintenance stories. The common solution is to stick to the release deadline and leave the remaining corrective stories to the next maintenance sprint, e.g. Subway Surfers, which are providing a whole set of seasonal updates in each month. The update includes a whole package of new graphic environment and numbers of new characters, which is a must-have in each sprint. Thus, according to their plan, event-related and seasonal features are as important as mandatory features in each maintenance sprint, which leads to the postponing of corrective features. And when the remaining corrective maintenance stories are still of high priority with numbers of user complaints, decision makers should consider inserting a minor update right after this maintenance sprint, or fitting them in the current maintenance sprint by extending a particular period of time. On the other hand, there are some teams that would rather finish all existing corrective maintenance stories first, e.g. Clash of Clan. The development team is providing a bigger time-box within which the team could accomplish a number of seasonal update including new warriors, new building units, new social features, and so on, which is not as relevant to each other as those of Subway Surfers. Therefore, it is more convenient for them to limit the numbers of seasonal features, leave them to next sprint and cope with as many as

corrective maintenance stories as possible.

The priority of corrective stories and event-related features in one sprint

When some event-related feature stories and corrective maintenance stories are selected for the next maintenance sprint, there are commonly concerns about which should the team cover first. It is also a twofold situation. Firstly, the sequence of selected user stories should be listed according to the calculated priority. Some teams that have enough developers and maintainers are able to divide the team into two sections covering respectively new features and maintenance. But that is not always possible when some team has fewer members. On the other hand, concerning different applications, particular emphasis of teams are various as well. For Subway Surfers, seasonal features are mandatory and are supposed to be accomplished in one piece first with all corrective maintenance stories to cover afterward. But for Clash of Clans, corrective maintenance stories and new features are of equal importance and will be prioritized strictly according to formula calculation.

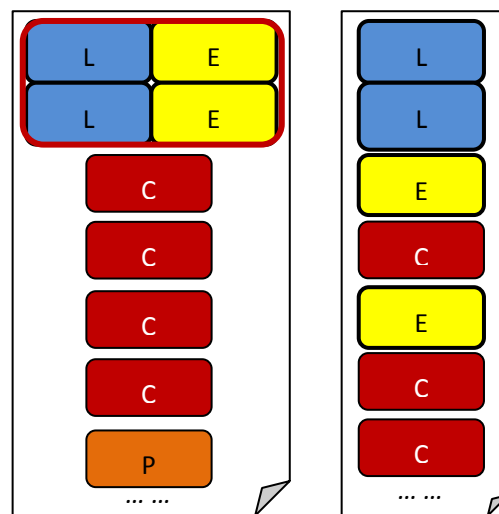


Figure 6.2 Two ways to compare corrective stories and seasonal features

Should perfective maintenance stories always go last?

It still depends on the actual priority of each user story. When perfective maintenance requests are numerous and seriously hindering users' regular gameplay or application usage, the priority of them will be higher than some corrective

maintenance stories. However, most likely teams and decision makers would prefer to cover at least some of the perfective maintenance stories only when corrective maintenance stories are not urgent enough and all seasonal and event-related features have been accomplished.

6.3. Constant Maintenance Model

When the maintenance sprint starts, the team would follow the instruction of the plan as much as possible. However, the plan is flexible in a way that everyday a stand-up meeting would be held to update progress and detect issues. When there are newly emerged bugs or other urgent perfective maintenance requirements retrieved from user feedbacks, a discussion on whether to change priorities of relevant stories or stick to the plan would be performed. At the end of the stand-up meeting each day, each team member should have a clear goal in mind for the work of that day and the progress situation of the whole group in current phase. The discussion would mainly be based on the three key factors to influence the priority, user expectation, business value and risks. According to the update pace, when the length of each maintenance sprint is short, the team would rather focus more on stability of the application by fixing as many existing bugs as possible and avoiding triggering new ones. When possible, some high-prioritized perfective maintenance stories should be included. When a costly new feature is brought ahead in priority list, the team should consider the next update a major one, which would most likely last longer than regular ones. Within the whole maintenance process, feedback retrieving and analysis, backlog management and user story classification and prioritization are constantly performed. Whenever needed, changes in plans would take place based on the approval of decision makers.

Similarly, as with the other two maintenance models, the first step of planning is also extracting maintenance user stories from user feedbacks and estimating. The methods are similar as well. Based on different types of maintenance stories and different user expected features, the backlog will be categorized as well. However,

for each maintenance sprint, when selecting maintenance stories and feature stories, there are various guidelines to follow according to the characters of applications and sometimes decision makers' preference. Generally, the guidelines of selecting user stories for the following maintenance sprint are listed as followed.

- Mandatory feature stories and adaptive maintenance stories should always be included in one of the earliest possible sprint and assigned highest priority.
- As many as possible corrective maintenance stories should be included.
- Pick some of the linear features and perfective maintenance stories of high priorities.
- If there is still time, pick excitors.

However, there are still exceptions for sure. And here are some general guidelines for some specific situations.

When the time-box is limited for all corrective maintenance stories, proceed as follows

As we choose to limit the duration of each maintenance sprint at two weeks (plus or minus two days), it is quite possible that the time-box is not able to contain all reported corrective maintenance stories. One of the solutions is to select the minimized number of other types of feature stories and maintenance stories. Of course, if there are adaptive maintenance stories and mandatories feature stories, they are still should be selected and prioritized high. Then the remaining room of the time-box should be filled in with corrective maintenance stories taking as many as possible. On the other hand, the other solution is to include only a limited number of corrective maintenance stories when decision makers consider that the rest of corrective maintenance stories are not urgent enough to deal with in this maintenance sprint. In this way, it means decision makers focus more on providing new application experiences to the users, compared to those who focus more on maintaining bugs.

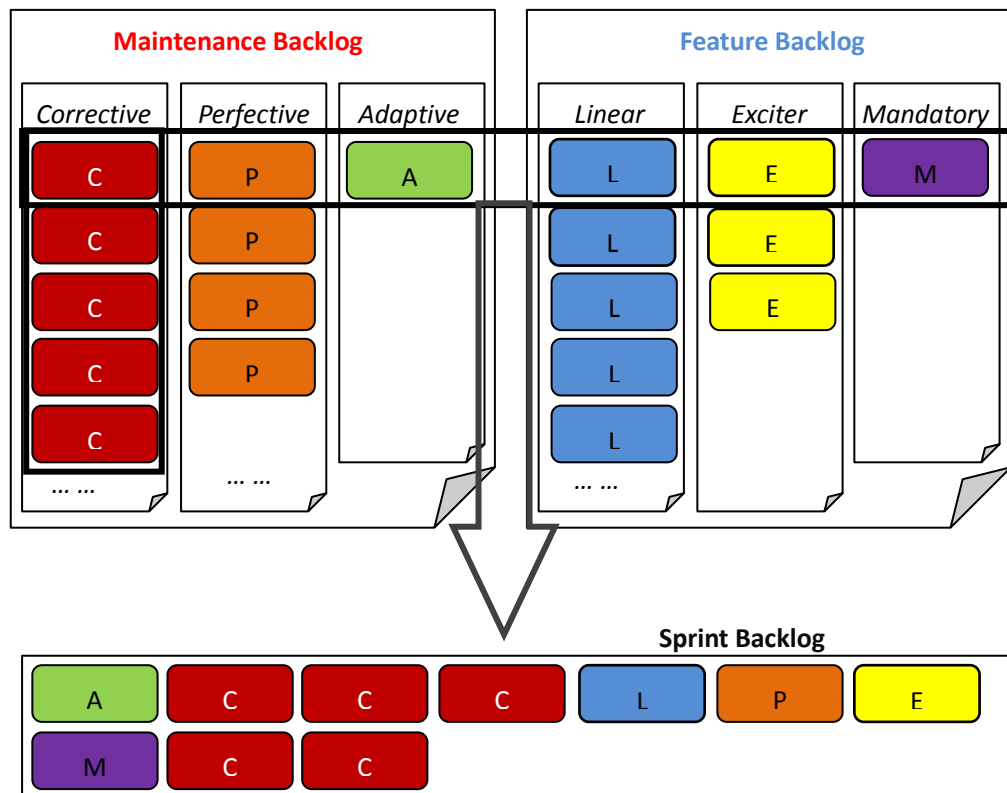


Figure 6.3 Selecting user stories in Constant Maintenance Model

It is always an option to make one sprint only containing corrective maintenance stories and perfective maintenance stories, when there are already major new features updated in the previous maintenance sprint. Depending on application types, it takes couple of maintenance sprints for users to get tired of existing features and expect new ones again. This period of time is the one in which the team and decision makers should try fixing bugs and make perfection for the application while planning for the next exciting new feature.

Update pace for new features

As far as we know, mandatory features are compulsory for the team to update as soon as possible with all possible effort. Most likely, most of the core mandatory features were previously covered in development phase. Even though there might be some mandatory features left for the first couple of maintenance sprints, the possibility of emerging new mandatory features is quite rare. It is not wise to use up all new ideas of features in backlog too fast. Thus, a continuous pace of updating

new linear features and exciter features is highly needed. Moreover, new ideas for features do not emerge all at once but gradually. One of the examples of setting the pace of it is presented in the following figure.

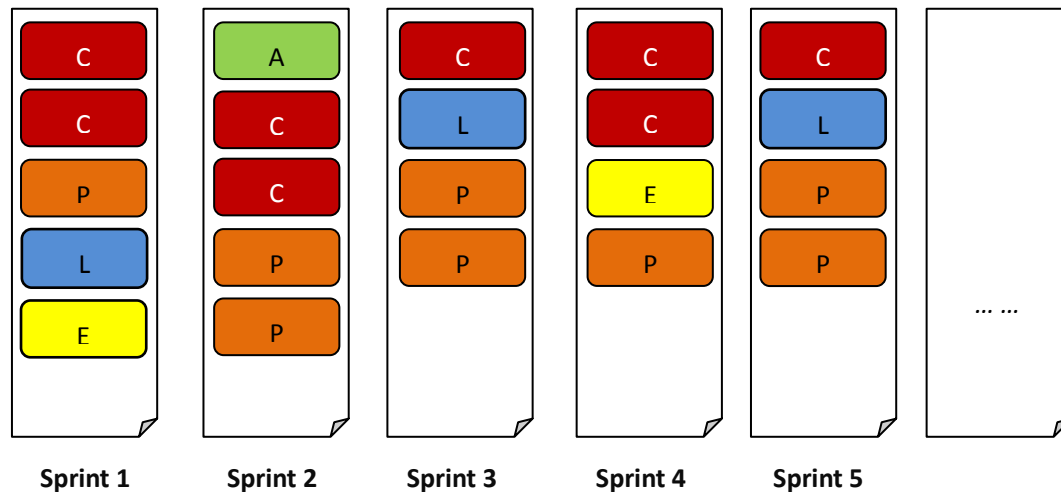


Figure 6.4 Setting pace for updating feature stories with Short Sprint

Depending on the estimated effort for each new feature story, the frequency varies. It is not compulsory to cover feature stories regularly in every sprint, but when it is possible updating new features each maintenance sprint it is still recommended to do so. On the other hand, when some features are firmly related to each other, they are commonly updated in one major release, which would most likely extend the sprint.

Adjusting time-box flexibly and minor sprint

Concerning actual applications, there are almost never any applications which could apply maintenance model by sticking to a strictly set time-box. Normally, the time-box is flexible concerning the selection of user stories in specific maintenance sprint and other customized reasons. However, there are situations when certain version of an update cannot satisfy a large amount of users or there are new potential bugs detected by the team after the release of the update in a short time, and a minor update is highly needed to fix bugs. The new corrective maintenance story will be prioritized to the highest. The minor update will be release as soon as possible before more users start to complain. Normally it takes two days to one week while sometimes it takes only a couple of hours.

6.4. Summary

Planning maintenance sprints is one of the key steps in performing an efficient agile maintenance for mobile application on IOS. A proper maintenance model selection and decent planning according to the maintenance model would simply result in good efficiency in delivering updates and organization in team management.

Regardless of which possible maintenance model the team and decision makers would adopt, some preparation steps are inevitable.

- Extracting user stories from feedbacks;
- Estimating story points for each user story;
- Setting priority for each user story (concerning user expectation, business value and risk);

By finishing the preparation steps, the team and decision makers would subsequently keep planning their first and following maintenance sprints based on the maintenance model they have chosen. Importantly, those preparation steps will be continuing through the whole maintenance process. As a result, the backlogs will be constantly updated and managed. In a Scrum-like maintenance process, the priority and user story selection will not be changed during the maintenance sprint. In Kanban maintenance process, even after the sprint starts the sprint backlog is changeable. However, when a user story is already worked on, it will be continuously proceeding.

Concerning choosing a suitable maintenance model to carry out proper maintenance plan, the guidelines are summarized in the following table.

Maintenance Models	Choosing Guidelines
Emergency-Oriented (e.g. Jishang Splendid Mahjong)	<ul style="list-style-type: none"> • Core mandatory features are sufficient; • Bugs are ignorable and performance is satisfactory enough with limited user complaints; • No linear features are needed; • Time or funds are limited; • The application is closing to the end of its lifecycle; • Team members are covering multiple applications, the others of which require much more efforts;
Event-Oriented (e.g. Subway)	<ul style="list-style-type: none"> • Normally Game Application;

Surfers, Clash of Clans)	<ul style="list-style-type: none"> • Game applications including various levels with increasing difficulties; • Game applications with themes that would vary from time to time; • Applications with customized themes to choose;
Constant (e.g. Facebook, Foursquare)	<ul style="list-style-type: none"> • Most applications other than games • Applications that contain huge amount of feature categories; • Applications that contain internet-related features more than standing-alone features; • Applications contain multiple social features;

Table 6.1 Maintenance Model Selection

Concerning different maintenance models, there are a number of guidelines when planning maintenance sprints. Relevant guidelines are presented in the following table.

	Emergency-Oriented	Event-Oriented	Constant
Concerned user story types	Covering almost Maintenance stories only.	Features & Maintenance	Features & Maintenance
Core idea	<ul style="list-style-type: none"> • Fixing corrective and adaptive maintenance stories when needed. • Covering perfective maintenance stories when time is left. 	<ul style="list-style-type: none"> • Updating event-related or seasonal features according to season calendar. • Covering adaptive maintenance stories and mandatory feature stories if any. • Containing potential corrective maintenance stories choosing as many as possible in one sprint. • Covering perfective maintenance stories when there is time. 	<ul style="list-style-type: none"> • Updating regularly in sprints of short period of time compared to Event-Oriented Maintenance Model. • Covering adaptive maintenance stories and mandatory feature stories if any. • Besides that evenly selecting different types of feature stories and maintenance stories.
Focus	Low risk, adaption and bug free. Maintain stability	Business value and continuous output of attractive features and perfection	Level user expectation, Adaption and Performance
Time-box Flexibility	Unregularly settled time point, not quite flexible in time-box.	Flexible in time-box when guaranteeing update content and pace.	Limited flexibility in time-box with a possibility in adjusting the time-box.
Priority setting	<ul style="list-style-type: none"> • Risk weight increased • User expectation weight increased 	<ul style="list-style-type: none"> • Apply high customized weight to Event-Related or seasonal features stories • Business value weight increased 	Generally stick to the calculated priority.

Update Pace	Random update pace. (All updates are relevantly minor.)	<ul style="list-style-type: none"> • Update in regular pace with seasonal and event-related feature ensured. Possibility to combine with Emergency – Oriented Model (e.g. Subway Surfers) • Update in flexible pace with sufficient amount of update content accomplished. (e.g. Clash of Clans) 	Generally the pace could be set ranging from 2 weeks to 20 days. Based on market strategies, product life cycle or other business-related factors, the pace could be adjusted accordingly.
--------------------	--	--	--

Table 6.2 Maintenance Models Guidelines Summary

Additionally, what we have to emphasize is that any type of maintenance model could only be considered as a brief framework that suits considerably better certain types of applications. For some cases, insisting on certain maintenance model which is seemingly not the recommended one is also possible. On the other hand, one single mobile application will somehow switch from one maintenance model into another when it emerges into certain phase of its life cycle under a bunch of complicated and somehow unpredictable circumstances.

7. Discussion

After presenting the agile maintenance models for mobile applications, there are still issues open for discussion. The aim of this discussion is to deliver more explicit information concerning certain specific topics.

7.1. Switching and Combining Maintenance Models

Based on the research on a number of existing mobile applications in the Apple app store, we have identified three major maintenance models in general, which have already been discussed respectively in details in previous chapters. On the other hand, there are a part of the sample applications adopting an ambiguous maintenance model, which is not able to be defined exactly as one of the typical maintenance models. However, further research on the update history of those samples shows that at certain point of the life cycle of some applications the adopted maintenance model might be switched from one to another as needed.

Taking one of the most popular action games on both IOS and Android platform, Temple Run, as an example, the change in maintenance models is obvious. The update history of Temple Run is presented in the following table.

V1.0	27.07.11	
V1.1	10.08.11	<ul style="list-style-type: none"> • Stats display bug fix (Corrective) • New Objectives to achieve (Linear)
V1.2	16.09.11	<ul style="list-style-type: none"> • Bug fixes (Corrective) • More characters (Linear) • More power ups (Linear) • More achievements (Linear) • Special offer of free coins (Exciter)
V1.3	11.10.11	<ul style="list-style-type: none"> • Bug fixes (Corrective) • Performance enhancement (Perfective) • New characters (Linear) • More Achievements (Linear) • Twitter score (Exciter) • More ways to earn free coins (Linear) • iPad support (Adaptive)
V1.4	19.11.11	<ul style="list-style-type: none"> • Bug fixes (Corrective) • Game Center support (Exciter) • New characters (Linear) • New achievement (Linear)

		• New wallpapers (Linear)
V1.4.1	03.12.11	• Bug fixes (Corrective)
V1.5	01.08.12	• Graphics enhancement for Retina (Perfective) • New power ups (Linear) • More achievements (Linear) • Bug fixes (Corrective) • Ability to enable and disable individual power ups (Exciter)
• V1.6	• 19.09.12	• Adaptive for iPhone 5 and IOS 6(Adaptive) • New social features (Exciter) • Bug fixes (Corrective)

Table 7.1 Update History of Temple Run³⁰

Based on the update history and update content details of Temple Run, it is quite obvious to observe the fact that after the version 1.4 there are no more regular updates after those of before. Before version 1.4, every maintenance update was released at a regular pace of one month, which contained only couple of days' differences. The content of maintenance was of similar categories, including corrective maintenance, linear new features, and sometimes exciter new features or perfective maintenance as alternatives. However, after version 1.4, the pace of maintenance was broken, while the following update contained only corrective maintenance and the maintenance sprint covered only roughly two weeks as well. Moreover, the next update released after the version 1.4.1 minor update took over half a year, though the update content was similar to the previous version. Additionally, the latest update version was released 50 days after previous version; containing only adaptive maintenance and corrective maintenance (the one exciter new feature was ignorable).

The update history shows that the maintenance model of Temple Run had switched from Process-Oriented Maintenance Model to Emergency-Oriented Model. The reason of model switching could be inferred as focus transferring to brand new version of Temple Run 2 as well as other derivative versions such as Temple Run: Brave and Temple Run: Oz. As a result of the limited resources and current stability and satisfactory functionality, the application will not be maintained continuously. On the other hand, another sample application which has also switched maintenance model from Process-Oriented Maintenance Model to Emergency-Oriented

³⁰ <https://itunes.apple.com/fi/app/temple-run/id420009108?mt=8>

Maintenance Model is Renaissance of Civilizations, which is an IOS game developed by a Chinese software company. The starting pace of update is similar to Temple Run. However, when the stability of the application was increased, the length of each maintenance sprint was extended from one month to around two months. Furthermore, after Version 2.1, there was only one adaptive maintenance update and another major update after 5 months. Based on interviews of CEO of this software company, the company has started to develop a brand new game application.

Therefore, based on the fact that the applications which have switched their maintenance models, we conclude that decision makers are able to switch existing maintenance model to a more efficient one. There are various factors that would affect the decision of changing maintenance models in order to better allocate effort, time and money for the best profits at certain point of the application life cycle.

On the other hand, combining maintenance models is another efficient way of dealing with emergency and organizing the maintenance process better. Mostly the possible way of combination is to add emergency-oriented updates up to the other maintenance models, which aims to cover urgent user requirements in the middle of one maintenance sprint when other user stories are not accomplished at that moment. After discussing with the team, decision maker is obliged to decide whether to deliver emergency update between regular updates, which might disturb the original plan of the project. Most of the applications applying Event-Oriented Maintenance Model or Constant Maintenance Model allow delivering minor update dealing with emergency.

7.2. Balance of Agility and Models

Regarding the characteristics of current mobile application maintenance and agile maintenance process, people are most likely swinging between the ideas of being more agile or disciplined, which is a universal issue in both software development and maintenance. Based on the discussion in this paper and former literature, the advantage of agile maintenance is obvious. Therefore, ‘How agile could or should we

be?’ became an obvious question. Based on the different agile maintenance models concerning mobile applications mentioned above, we could easily assume that it is possible that there is one perfect model for every single one application in their maintenance phase when the team and clients are quite confident in classifying the application and the prospective of it. However, the situation and evolution of market and technology is to large extent unpredictable. Thus, the maintenance model is meant to be shifted accordingly and in a more agile and customized way.

Firstly, the fact that mostly certain maintenance model is indeed more suitable for certain types of applications than others should be emphasized. Some application does not have to be maintained every two weeks as there is neither space for exciter features or perfection nor bugs to fix all the time. Constant maintenance may result in a huge waste of time, funds and efforts. In contrast, some applications are obliged to be maintained as the features are complex and versatile with huge prospective to enhance. Poor maintenance either causes the decline in stability of the system or in the attraction of exciting features, which would both result in dissatisfaction of users.

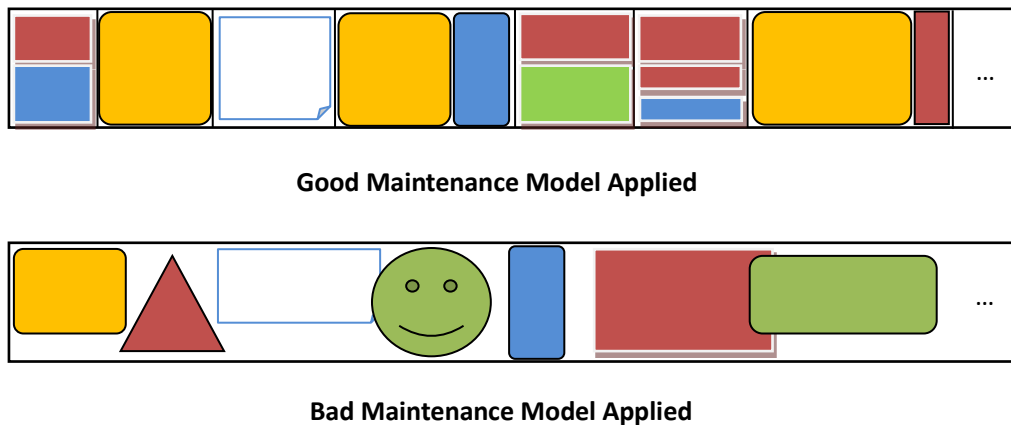


Figure 7.1 Good and Bad Maintenance Model Application

Therefore, the key idea of applying efficient and profitable agile maintenance models is to keep being agile in a framework of discipline. Figure 7.1 is a metaphor of differentiating good and bad maintenance model applied. In the figure, blocks of different colors are representing different types of stories, which are the same as the ones in other figures in previous chapters. The different shapes in the second figure

are to emphasize the time wasting consequence of bad planning. In the good maintenance application example, all maintenance sprints are planned properly and delivered at a decent pace. Though the length of each sprint is slightly different depending on the estimation of each story, the team manages to have clear deadlines and completed update content. On the contrary, bad maintenance planning would most likely results in the second example in Figure 7.1 with huge amount of time wasted and uncompleted features delivered.

A good maintenance model does not set strict deadline for updates but it keeps tracing high prioritized requirements and achieves them in time. When no emergency issues are pending, it is good to include most valuable features or maintenance stories and accomplish those within a sprint of reasonable length, and deal with emergency as soon as possible when it detected. The process flows smoothly when the system is kept steady and runnable as always with little time wasted and maximized value realized. Oppositely, a bad maintenance model could be considered as too agile to be efficient and profitable. Leaving critical bugs for too long, inability to predict market trend and following others blindly, changing without thorough consideration, and so on are the superficial behaviors. Losing team morale, losing motivation and direction, wasting time, making little or minus profit, etc. are the obvious consequences.

Another decent, beneficial but challenging way to apply an agile maintenance model is combining the models to make the maximized use of them. The process is to follow the pace of constant maintenance model in general when setting maintenance sprint duration a bit longer. When running across seasonal points or events, the team will release relevant seasonal or event updates. Additionally, to deal with emergency, at any time of a maintenance sprint an emergency update is allowed to be released when the sprint timeline will be deliberately delayed. However, combination of models requires an experienced management from the project owner and other decision makers, as well as a highly self-organized, experienced team.

On the other hand, within one typical application project, the maintenance model

might not be constant throughout the whole life cycle. For instance, when an application went through a mature development and maintenance process, the stability of it will be unquestionable. When not much space is left for exciter features or further perfection, the team would consider switching original Constant maintenance model into Event-Oriented maintenance model. On the other hand, a project would also switch into Constant maintenance model when certain technology or product provokes a prevailing trend for mobile application and the market is unpredictably promising. Thus, a clear mind in market trend and creativity is also an important capability for decision makers when they are on the edge of changing models.

7.3. The Role of the Decision Maker

Based on the studies in this paper, it is obvious that the decision maker of a mobile application project is playing a critical role in the whole development and maintenance phase. Thus the obligations and rights of this critical decision maker role have tremendous and direct impact on the whole project in all possible ways.

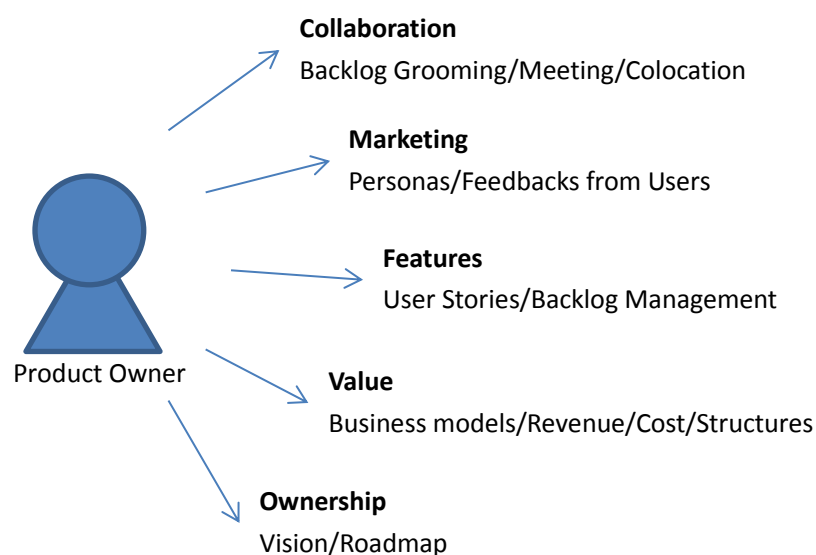


Figure 7.2 Role of Product Owner in Scrum

To discuss the role and responsibility of the decision maker in agile maintenance

process, reviewing the role of product owner in Scrum is compulsory. Product owner was defined as the person responsible for managing the backlog so as to maximize the value of the project [Schwaber, 2004]. To further define the role, product owner is a person who is to large extent self-involved in the whole development phase, has an explicit vision of the target product, proposes requirements to the team and makes sure that what the team develops is what he/she wants, and manages backlogs to optimize the returning value. The main aim of adopting agile principle in software development is to satisfy the customers by delivering valuable software in early stage and continuously. The product owner stands in a position which is so important that all vital decisions are supposed to be made by only product owner, which means that product owners will single-handedly direct the whole project into their own expectations despite of all objective circumstances. Based on the description of Pichler [2010_2], the main obligations of a product owner will cover five aspects listed in Figure 7.2.

On the other hand in Kanban method, the role of product owner is not compulsorily prescribed [Kniberg et al., 2009], as Kanban is neither a development process nor a project management methodology, but a change method [Roock, 2011]. However, the discussion on whether there should be a product owner or similar role in Kanban process is still continuing. Based on the study of Roock [2011], there is a typical situation of chaos in requirements prioritization when developers and testers take care of prioritization by themselves. The beginning of the process is decent without product owner when at some point the decisions of setting prioritization will go randomly by shouting out loud. Therefore, someone needs to collect requirements and prioritize them, contact stakeholders, create a product vision, ensure implementation, and so on, which is basically the product owner's role. Thus it is recommended to set a product owner role for the team as a connection of the team and other stakeholders. Additionally, it is a proper to settle conflicts in all aspects by adopting product owner handling all the tasks. The same situation concerns software maintenance phase. It is even more important to have a decision maker role in mobile

application process, as changes in business models, maintenance models, and market trends are even more spontaneous. Thus a quick response to changes by following decisions from decision makers would be necessary, which also accordingly requires the decision maker to have the suitable qualification for the obligation.

Moreover, there are general doubts concerning whether there should be more product owner roles in one project team in order to share the responsibility and burden. The studies concerning the pros and cons of a product owner team include [Pichler, 2010_3] [Cottmeyer, 2011] [Löffler, 2011] [Moe, 2009] [Paasivaara, 2012]. Based on the studies, the advantages of having a product owner team is quite obvious, including sharing responsibility, teamwork, availability extension, diverse experiences, and optimized environment. However, the actual need of containing multiple product owners as decision makers is challenged as well [Brodzinski, 2010]. The main defects of having multiple decision makers in agile team is that the chaotic organization and disagreement amongst decision makers could bring disaster to the whole project when what the team needs the most is quick decisions coping with randomly emerging changes and rapidly changing market trends. Concerning agile mobile application maintenance, the recommended number of decision makers is two at the maximum, which cover issues from new feature user stories and maintenance stories respectively. For small-scaled agile development teams, a single decision maker will be sufficient for running the whole project and making critical decisions when the product owner is well qualified with characteristics such as knowing the product vision and the domain well, being a good communicator, engaging customers and users, and so on. Otherwise, an additional decision maker will be needed to help out, or simply to take the position of the previous.

8. Conclusion

Based on the study on literature and the research on existing popular mobile applications, a concise agile maintenance process for mobile application is proposed. Furthermore as a critical step in the agile maintenance process, three agile mobile application maintenance models are constructed. General guidelines were provided for choosing and applying a certain maintenance model for real-life projects. Based on the principles of agile development and software maintenance process, the process of each maintenance model is described concisely. By adopting mobile application maintenance models, as the guidelines are limited the team and decision makers will embrace agile development to the most. However, the idea is to create an efficient harmonic maintenance flow to maximize the business value and user satisfaction, and additionally minimize all possible cost and risk during the process. The core of mobile application maintenance model is prioritizing feature user stories and maintenance user stories in an efficient and beneficial way so that the features and maintenance updates which provides the most business value or user satisfaction will be covered firstly. It also means that on certain occasions, business value and user satisfaction will not be fulfilled at same time. Moreover, even though when some features or maintenances are providing tremendous business value and user satisfaction, the decision will be under serious concerns when the risks of accomplishing those functionalities are rather high. Thus, concerning the characteristics of different applications and various user groups the applications targeting, a balance between business value, user satisfaction or expectation and risks should be found.

Even when the team and decision makers are explicitly clear with the maintenance process, there are always expected the market trends which will change decisions dramatically. However, that is exactly why maintenance process is obliged to be agile and flow-like. In this study, by providing the minimum guidelines concerning maintenance models, we are also delivering the ideas of creating a working flow for mobile application maintenance. By applying maintenance models,

the team and decision makers will focus on minimum impacting factors and more on the valuable part of maintenance work.

Despite the contribution of this thesis, there are still limitations as well. Firstly, the range of the general research direction is wide with multiple branches. The whole structure of agile maintenance is still discussed and formed in a limited way. This study is only focusing on mobile applications maintenance, not covering circumstances in other platforms and leaving also details uncovered. In addition, only limited number of cases has been observed due to the huge quantity of the total amount of applications. Furthermore, the study is based on mostly literature and observations of existing cases and provides theoretical description, but it lacks validation and verification that proves the benefits of using maintenance models in practices. On the other hand, the study is not a holistic theoretical solution as a silver bullet when more details should be covered concerning the different characteristics of mobile applications of different categories and market regions. Based on the limitation, the possible future research will focus on verification and improvement. The whole mobile application maintenance models model requires perfection in many ways including extra market factors analysis, region and culture factors, details of the maintenance process and time allocation in details.

References

- [Abrahamsson et al., 2002] Pekka Abrahamsson, Outi Salo, Jussi Ronkainen, & Juhani Warsta. Agile Software Development Methods—Review and Analysis. VTT Publication. 2002.
- [Abrahamsson et al., 2004] Pekka Abrahamsson, Antti Hanhineva, Hanna Hulkko, Tuomas Ihme, Juho Jäälinoja, Mikko Korkala, Juha Koskela, Pekka Kyllönen, and Outi Salo. Mobile-D: an agile approach for mobile application development. OOPSLA'04, Oct. 24–28, 2004, (pp. 174-175).
- [Abrahamsson, 2005] Pekka Abrahamsson. Mobile software development - the business opportunity of today. 2005. Proceedings of the International Conference on Software Development, (pp. 20-23). Reykjavik.
- [Abrahamsson et al., 2012] Pekka Abrahamsson, Antti Hanhineva, Hanna Hulkko, Tuomas Ihme, Juho Jäälinoja, Mikko Korkala, Juha Koskela, Pekka Kyllönen, & Outi Salo. Mobile-D: An Agile Approach for Mobile Application Development. OOPSLA'04, Oct. 24–28, 2004, Vancouver, British Columbia, Canada. ACM 1-58113-833-4/04/0010.
- [Anderson, 2003] David Anderson. Agile Management for Software Engineering: Applying the Theory of Constraints for Business Results. Prentice Hall. 2003
- [Arnold et al., 1993] Robert S. Arnold, & Shawn A. Bohner. Impact Analysis – Towards a Framework of comparison. Software Maintenance 1993 CSM-93 Proceeding Conference. 1993.
- [Arrayent, 2012] Arrayent.com. White paper: On selecting an Optimal Mobile Application Business Model. 2012. <http://www.arrayent.com/pdfs/TheOptimalMobileAppBusinessModels.Arrayent.2012-01-26.pdf>
- [Barua et al., 1995] Anitesh Barua, Charles H. Kriebel, and Tridas Mukhopadhyay. Information Technologies and Business Value: An Analytic and Empirical Investigation. Information Systems Research. 1995. Vol. 6.
- [Beck, 2000] Kent Beck. Extreme Programming Explained: Embrace Change. Addison-Wesley Professional. 2000.

- [Beck et al., 2001] Kent Beck, Mike Beedle, Arie van Bennekum, Alistair, Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, Jon Kern, Brian Marick, Robert C. Martin, Steve Mellor, Ken Schwaber, Jeff Sutherland, & Dave Thomas. Manifesto for Agile Software Development. 2001. <http://agilemanifesto.org/>
- [Beck et al., 2005] Kent Beck, & Cynthia Andres. Getting Started with Xp: Toe Dipping, Racing Dives and Cannonballs. Three Rivers Institution. 2005.
- [Beedle et al., 2001] Mike Beedle, Kent Beck, Arie van Bennekum, Alistair, Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, Jon Kern, Brian Marick, Robert C. Martin, Steve Mellor, Ken Schwaber, Jeff Sutherland, & Dave Thomas. Principles Behind the Agile Manifesto. 2001 <http://agilemanifesto.org/principles.html>
- [Bennett et al., 2000] Keith H. Bennett, & Vaclav T. Rajlich. Software Maintenance and Evolution: A Roadmap. ICSE 00 Proceedings of the Conference on the Future of Software Engineering. ACM, New York. 2000. <http://dl.acm.org/citation.cfm?id=336534>
- [Bettini et al., 2009] C. Bettini, S. Jajodia, P. Samarati, & S.X.Wang. Privacy in Location-Based Applications. Research Issues and Emerging Trends. Springer. 2009.
- [Boehm, 2002] Boehm, B. Get Ready for Agile Methods, with Care. 2002. Computer , 35 (1), 64-69.
- [Brodzinski, 2010] Pawel Brodzinski. Why having Multiple Product Owners is a bad idea. May 2010. <http://brodzinski.com/2010/05/multiple-product-owners-bad-idea.html>
- [Claybrook, 2012] Bill Claybrook. Mobile cloud apps vs. native apps: The developer's perspective. 2012. <http://searchcloudapplications.techtarget.com/feature/Mobile-cloud-apps-vs-native-apps-The-developers-perspective>.
- [Claybrook, 2012_2] Bill Claybrook. Mobile cloud trends: Apps let enterprises handle the risks of cloud computing. 2012. <http://searchcloudapplications>.

techtargget.com/feature/Mobile-cloud-trends-Apps-let-enterprises-handle-the-risks-of-cloud-computing

[Clegg et al., 1994] Dai Clegg, & Richard Baker. Case-Method Fast Track: A Radical Approach. Addison-Wesley Longman Publishing Co. 1994, <http://dl.acm.org/citation.cfm?id=561543>

[Coad et al., 1999] Peter Coad, Jeff de Luca & Eric Lefebvre. Java Modeling In Color With UML: Enterprise Components and Process. Prentice Hall. 1999.

[Cohn, 2004] Mike Cohn. User Stories Applied for Agile Software Development. Addison-Wesley. 2004. <http://www.mountaingoatsoftware.com/books/user-stories-applied>

[Cohn, 2010] Mike Cohn. Prioritizing Your Product Backlog. Mountain Goat Software. Jun 8, 2010. <http://www.mountaingoatsoftware.com/presentations/prioritizing-your-product-backlog>.

[Copeland, 2001] Lee Copeland. Extreme Programming. 2001. http://www.computerworld.com/s/article/66192/Extreme_Programming?taxonomyId=063

[Cottmeyer, 2011] Mike Cottmeyer. Why a Product Owner Team. Mar. 28. 2011. <http://www.leadingagile.com/2011/03/why-a-product-owner-team/>

[Crockford, 1986] Neil Crockford. An Introduction to Risk Management (2nd Edition). Woodhead-Faulkner. Cambridge, UK. 1986.

[Deemer et al., 2006] Pete Deemer, & Gabrielle Benefield. Scrum Primer. Yahoo.2006. <http://www.scrumalliance.org/resources/339>

[DeGrace et al., 1990] Peter DeGrace, & Leslie Hulet Stahl. Wicked Problems, Righteous Solutions:aCatalogue of Modern Software Engineering Paradigms. Yourdon Press. 1990

[De Vere, 2012] Kathleen De Vere. Supercell's free-to-play titles grossing \$15M a month on iOS. Oct. 2012. <http://www.insidemobileapps.com/2012/10/09/supercells-free-to-play-titles-grossing-15m-a-month-on-ios/>

[Dong, 2013] Wen Dong. Downloads hit a record of 25 billion during the 10 years of

- iTunes. 2013. <http://music.yule.sohu.com/20130207/n365779644.shtml>
- [Eddy, 2012] Nathan Eddy. Location-Based Applications Popular, Despite Privacy Concerns: ISACA. 04.05.2012. <http://www.eweek.com/c/a/Mobile-and-Wireless/LocationBased-Applications-Popular-Despite-Privacy-Concerns-ISACA-181357/>
- [Edmonds, 1974] E.A Edmonds. A Process for the Development of Software for Nontechnical Users as an Adaptive System. *General Systems* 19: 215-18.
- [Erdil et al., 2003] Kagan Erdil, Emily Finn, Kevin Keating, Jay Meattle, Sunyoung Park, & Deborah Yoon. Software Maintenance as Part of the Software Life Cycle. *Hepguru.com*.2003.http://hepguru.com/maintenance/Final_121603_v6.pdf
- [Fling, 2009] Brian Fling. *Mobile Design and Development Practical concepts and techniques for creating mobile sites and web apps*. O'Reilly Media. 2009.
- [Gasimov et al., 2010] Anar Gasimov, Chuan-Hoo Tan, Chee Wei Phang & Juliana Sutanto. Visiting Mobile Application Development: What How and Where.2010 Ninth International Conference on Mobile Business. 2010.
- [Gowans, 2001] David Gowans. WML (Wireless Markup Language) : An Introduction.2001. <http://www.sitepoint.com/markup-language-introduction/>
- [Herbert, 2011] Chris Herbert. In-App Purchase Revenue Growing as Developers Adopt “Freemium” Model. 2011. <http://www.macstories.net/news/in-app-purchase-revenue-growing-as-developers-adopt-freemium-model/>
- [Holler, 2006] Robert Holler. *Mobile Application Development: A Natural fit with Agile Methodologies*. 2006. VisionOne LLC.
- [Hughes, 2012] Adrian Kinsley-Hughes. China expected to overtake U.S. to become world's largest iOS and Android market. 2013. <http://www.zdnet.com/china-expected-to-overtake-u-s-to-become-worlds-largest-ios-and-android-market-7000011447/>.
- [Hunt et al., 2008] Bob Hunt, Bryn Turner, & Karen McRitchie. Software Maintenance Implications on Cost and Schedule. *Aerospace Conference, 2008 IEEE*. 2008. <http://ieeexplore.ieee.org/xpl/articleDetails.jsp?tp=&arnumber=4526688&contentType=Conference+Publications&queryText%3Dsoftware+maitenan>

ce+implications+on+cost+and+schedule

[IEEE, 1998] IEEE Computer Society. IEEE standard for Software Maintenance. 1998. ISBN 0-7381-0336-5.

[Jeffries, 1999] Ronald E. Jeffries. What is Extreme Programming? 1999. <http://xprogramming.com/what-is-extreme-programming/>

[Judy et al., 2008] Ken H. Judy & Ilio Krumins-Beens. Great Scrums Need Great Product Owners: Unbounded Collaboration and Collective Product Ownership. Proceedings of the 41st Hawaii International Conference on System Sciences. 2008.

[Kagan et al, 2003] Kagan Erdil, Emily Finn, Kevin Keating, Jay Meattle, Sunyoung Park, & Deborah Yoon. Software Maintenance as Part of the Software Life Cycle. University of Tuft. 2003. http://hepguru.com/maintenance/Final_121603_v6.pdf.

[Kajko-Mattsson et al., 2009] Mira Kajko-Mattsson and Jaana Nyfjord. A Model of Agile Evolution and Maintenance Process. The 42nd Hawaii International Conference on System Sciences. 2009

[Kano et al., 1984] Noriaki Kano, Nobuhiku Seraku, Fumio Takahashi, & Shinichi Tsuji. "Attractive quality and must-be quality" (in Japanese). Journal of the Japanese Society for Quality Control 14 (2): 39–48. April 1984. ISSN 0386-8230

[Karlsson, 2009] Goran Karlsson. 8 Characteristics of Good User Requirements. 2009. <http://www.slideshare.net/guest24d72f/8-characteristics-of-good-user-requirements-presentation>.

[Kniberg, 2006] Henrik Kniberg. Scrum and XP from the Trenches. Crisp.se. 2006. <http://www.crisp.se/henrik.kniberg/ScrumAndXpFromTheTrenches.pdf>

[Kniberg et al., 2009] Henrik Kniberg, & Mattias Skarin. Kanban and Scrum making the most of Both. InfoQueue. 2009. <http://www.infoq.com/resource/minibooks/kanban-scrum-minibook/en/pdf/KanbanAndScrumInfoQVersionFINAL.pdf>

[Koekkoek, 2011] Hendrik Koekkoek. Distimo Releases Full Year 2011 Publication. http://www.distimo.com/blog/2011_12_distimo-releases-full-year-2011-publicatio

n/. 2011

- [Koskinen, 2003] Jussi Koskinen. Software Maintenance Cost. 2003. University of Jyväskylä. <http://www.cs.jyu.fi/~koskinen/smcosts.htm>.
- [Larman, 2003] Craig Larman & Victor R. Basili. Iterative and incremental developments: a brief history. IEEE Computer Society. 2003. 0018-9162. <http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=1204375>
- [Lim, 2006] Andrew Lim. Candybar Phones. 2006. <http://reviews.cnet.co.uk/mobile-phones/candybar-phones-49273835/>
- [Löffler, 2011] Marc Löffler. 5 Reasons Why a Product Owner Team Might Be a Good Idea. Oct. 13. 2011. <http://blog.scrumphony.com/2011/10/5-reasons-why-a-product-owner-team-might-be-a-good-idea/>
- [McCormick, 2012] Zach McCormick, & Douglas C. Schmidt. Data Synchronization Models in Mobile Application Design, proceedings of the Model Languages of Programs (PloP) 2012 conference. 2012.
- [Martin et al., 1983] James Martin, & Carma L. McClure. Software Maintenance: The Problem and its Solution. Prentice Hall. 1983.
- [Mello, 2012] Moacyr Cardoso de Mello Filho. Agile processes for the maintenance cycle: A smarter work cycle for a Smarter Planet. 2012. <http://www.ibm.com/developerworks/rational/library/agile-processes-maintenance-cycle-pdf.pdf>
- [Miller, 2013] Ted Miller. App Store Tops 40 Billion Downloads with Almost Half in 2012. <http://www.apple.com/pr/library/2013/01/07App-Store-Tops-40-Billion-Downloads-with-Almost-Half-in-2012.html>. 2013.
- [MobiThinking, 2012] Global mobile statistics 2012 Home: all the latest stats on mobile Web, apps, marketing, advertising, subscribers, and trends...<http://mobithinking.com/mobile-marketing-tools/latest-mobile-stats>
- [Moe, 2009] Nils Brede Moe, Torgeir Dingsøy, & Øyvind Kvangardsnes. Understanding Shared Leadership in Agile Development: A Case Study. Proceedings of the 42nd Hawaii International Conference on System Sciences.

2009.

[Mowble, 2013] The Prophecies for App Development and Downloads in 2013. 2013. <http://www.mowble.com/the-prophecies-for-app-development-and-downloads-in-2013.html>.

[Mukhija, 2003] Arun Mukhija. Estimating Software Maintenance. Requirements Engineering Research Group, University of Zurich. 2003. http://www.ifi.uzh.ch/req/courses/seminar_ws02/reports/Seminar_9.pdf

[Mundy, 2013] Jon Mundy. iOS App Store hits 800,000 apps. Mar. 2013. <http://www.trustedreviews.com/news/ios-app-store-hits-800-000-apps>

[Neumann, 1997] Peter. G. Neumann. UK and Y2K: \$50 billion. 1997. <http://catless.ncl.ac.uk/Risks/19.07.html#subj6.1>

[Newton, 2010] Chris Newton. Software Maintenance. 09.01.2010. <http://clarityincode.com/software-maintenance/>

[Nordgard et al., 2003] D.E. Nordgard, J. Heggset and E. Ostgulen. Handling Maintenance Priorities using Multi Criteria decision making. IEEE Bologna PowerTech Conference. 2003.

[Ogg, 2011] Erica Ogg. By the numbers: Mobile apps in 2011. 2011. <http://gigaom.com/2011/12/30/by-the-numbers-mobile-apps-in-2011/>

[Paasivaara, 2012] Maria Paasivaara, Ville T. Heikkilä & Casper Lassenius. Experiences in Scaling the Product Owner Role in Large-Scale Globally Distributed Scrum. 2012 IEEE Seventh International Conference on Global Software Engineering. 2012.

[Paukkunen, 1999] Mintunkukka Paukkunen. Wireless Application Protocol. 1999. http://www.tml.tkk.fi/Studies/Tik-110.300/1998/Essays/wap_2.html

[Pichler, 2010] Roman Pichler. Agile Product Management with Scrum, Creating Products that Customers Love. Addison– Wesley. 2010. <http://www.romanpichler.com/publications/>

[Pichler, 2010_2] Roman Pichler. The Product Owner on one Page. Nov. 01. 2010. <http://www.romanpichler.com/blog/roles/one-page-product-owner/>

- [Pichler, 2010_3] Roman Pichler. Scaling the Product Owner. Jan.25. 2010.
<http://www.romanpichler.com/blog/roles/scaling-the-product-owner/>
- [Pino et al., 2011] Francisco J. Pino, Francisco Ruiz, Felix Garcia and Mario Piattini. A Software Maintenance Methodology for Small Organizations: Agile_MANTEMA. Journal of Software Maintenance and Evolution: Research and Practice. 2011.
- [Poole et al., 2001] Poole, C. J., & Huisman, J. W. (2001). Using extreme programming in a maintenance environment. IEEE Software, 18(6), 42-50.
- [Poppendieck et al., 2003] Mary Poppendieck & Tom Poppendieck. Lean Software Development: An Agile Toolkit. Addison-Wesley Professional. 2003.
- [Prakash, 2010] GouriPrakash. Achieving Agility in Adaptive and Perfective Software Maintenance. 14th European Conference on Software Maintenance and Reengineering. 2010. <http://www.computer.org/portal/web/csd1/doi/10.1109/CSMR.2010.23>
- [Pressman, 2005] Roger S. Pressman. Software Engineering: A Practitioner's Approaches (Sixth edition). McGraw– Hill. 2005. http://highered.mcgraw-hill.com/sites/0072853182/information_center_view0/
- [Racheva et al., 2010] Zornitza Racheva, Maya Daneva, Klaas Sikkels, Roel Wieringa and Andrea Herrmann. Do We Know Enough about Requirements Prioritization in Agile Projects: Insights from a Case Study. 18th IEEE International Requirements Engineering Conference. 2010.
- [Rajkumar, 2009] J.B. Rajkumar. Automated regression testing Challenges in Agile environment. 2009. <http://www.softwaretestinghelp.com/automated-regression-testing-challenges-in-agile-testing-environment/>
- [Ren et al., 2011] Yongchang Ren, Tao Xing, Xiaoji Chen, & Xuguang Chai. Research on Software Maintenance Cost of Influence Factor Analysis and Estimation Method. IEEE. 2011.
- [Reisinger, 2013] Don Reisinger. Apple App Store hits 40 billion downloads; 20 billion in 2012, alone. Jan. 2013. http://news.cnet.com/8301-13579_3-57562400-

- 37/apple-app-store-hits-40-billion-downloads-20-billion-in-2012-alone/
- [Rico, 2008] David F. Rico. Agile Methods and Software Maintenance. 2008.
<http://davidfrico.com/roi-online-f.htm>
- [Roman, 2010] Ernan Roman. Voice-of-the-Customer Marketing: A Revolutionary 5-Step Process to Create Customers Who Care, Spend, and Stay. Ernan Roman Direct Marketing Corp. 2010
- [Roock, 2011] Arne Roock. Do we need a Product Owner in Kanban? Aug. 2011.
<http://www.software-kanban.de/2011/08/do-we-need-product-owner-in-kanban.html>
- [Rosenberg, 2013] Jamie Rosenberg. Google Play hits 25 billion downloads. 2012.
<http://officialandroid.blogspot.fi/2012/09/google-play-hits-25-billion-downloads.html>
- [Russell, 2012] Jon Russell. China will be the world's biggest app market, and other predictions for Asia in 2012. 2012. <http://thenextweb.com/asia/2012/01/13/china-will-be-the-worlds-biggest-app-market-and-other-predictions-for-asia-in-2012/>
- [Salo, 2006] Salo, O. Enabling Software Process Improvement in Agile Software Development Teams and Organisations. 2006. Helsinki: VTT.
- [Schwaber, 2004] Ken Schwaber. Agile Project Management with Scrum. Microsoft Press. 2004. <http://www.microsoft.com/learning/en/us/book.aspx?id=6916&locale=en-us>
- [Sehlhorst, 2007] Scott Sehlhorst. Agile Development and Software Maintenance Cost. Feb. 2007. <http://tynerblain.com/blog/2007/02/28/agile-development-roi-2/>
- [Sheridan, 2012] Trevor Sheridan. iOS 6 App Store Adds Version History To Each App. 2012. <http://applenapps.com/ios/ios-6-app-store-adds-version-history-to-each-app.html>.
- [Smith, 2013] Robert Smith. State of the Mobile Enterprise – Appcelerator Q1 2013 Mobile Enterprise Report. 2013. <http://thinkmobile.appcelerator.com/resource-center/bid/262194/Q1-2013-State-of-the-Mobile-Enterprise-Appcelerator-Surveys>

- [Spataru, 2010] Andrei Cristian Spataru. Agile Development methods for Mobile Application. 2010. School of Informatics University of Edinburgh.
- [Stapleton, 1997] Jennifer Stapleton. DSDM: Dynamic Systems Development Method. Addison-Wesley. 1997.
- [Svensson et al., 2005] Harald Svensson & Martin Host. Introducing an Agile Process in a Software Maintenance and Evolution Organization. 2005. Proceedings of the Ninth European Conference on Software Maintenance and Reengineering (CSMR'05)
- [Szalvay, 2004] Victor Szalvay. An Introduction to Agile Software Development. Danube Technology. 2004. http://www.danube.com/docs/Intro_to_Agile.pdf
- [Takeuchi et al., 1986] Hirotaka Takeuchi, & Ikujiro Nonaka. The New Product Development Game. Harvard Business Review. 1986
- [Van Vliet, 2000] Hans Van Vliet. Software Engineering: Principles and Practices. 2nd Edition. John Wiley & Sons, West Sussex, England. 2000
- [Vannieuwenborg et al., 2012] Frederic Vannieuwenborg, Laurent Mainil, Sofie Verbrugge, Mario Pickavet, & Didier Colle. Business models for the mobile application market from a developer's viewpoint. 2012. 16th International Conference on Intelligence in Next Generation Networks.
- [Viken, 2009] Alexander Viken. The history of Personal Digital Assistants 1980 – 2000. 2009. <http://agilemobility.net/2009/04/the-history-of-personal-digital-assistants1/>
- [XinhuaNet, 2013] Which apps will win the market in 2013. http://news.xinhuanet.com/info/2013-03/18/c_132241426.htm
- [Ylimannela, 2012] Ville Ylimannela. A Model for Risk Management in Agile Software Development. Communications of Cloud Software. 2012.

Appendix 1

List of sample mobile applications

No.	App Name	Category	Region	Description	Model
1	Temple Run	Action Game	Universal	Action game. Swiping to avoid dangers and keep running.	C -> Em
2	Subway Surfers	Action Game	Universal	Action game. Similar to Temple Run	Ev + Em
3	Clash of Clans	Strategy Game	Universal	Strategy game. Building village and war against others	Ev
4	Carrot Fantasy (保卫萝卜)	Strategy Game	China	Strategy game. Cartoon style tower defending	C + Em
5	Fruit Ninja HD Free	Action Game	Universal	Action Game. Slicing fruits and avoiding bombs.	C
6	Ninja Jump	Adventure Game	Universal	Adventure Game. Avoid obstacles and keep running up.	Em
7	Dark Runner	Adventure Game	Universal	Adventure Game. Avoid	Em
8	Room break	Adventure Game	Universal	Adventure Game. Find clues and solve mystery	Em
9	Angry Bird Rio HD	Arcade Game	Universal	Arcade Game. Use birds as weapon to destroy buildings to pass levels.	C + Em -> Em
10	Doodle Jump HD	Arcade Game	Universal	Arcade Game. Tilt device to handle jumping direction avoiding falling.	C -> Ev + Em
11	Fishing Joy HD (捕鱼达人)	Arcade Game	China	Arcade Game. Fishing game.	Ev + Em
12	Fishing King	Arcade Game	Universal	Arcade Game. Fishing Game.	Em
13	Word with Friends HD Free	Board Game	Universal	Board Game. Scramble game with social feature.	C + Em
14	Jishang Splendid Mahjong (极上豪华麻将)	Board Game	China	Board Game. Traditional Mahjong Game.	Em
15	Ace Hearts Deluxe HD	Card Game	Universal	Card Game. Windows old Heart game with characters.	Em
16	Three Kingdoms Kill (三国杀)	Card Game	China	Card Game. Chinese hot board game transplanted on IOS.	Em
17	Bookworm HD	Word Game	Universal	Word Game. Build up words by selecting letters in maze.	Em
18	I Love Crossword (我爱填字)	Word Game	China	Word Game. Crossword game in Chinese	Em
19	Sanguo Gomoku (三国五子棋)	Trivia Game	China	Trivia Game. Traditional Go game with Sanguo characters	Em
20	4 Pics 1 Word	Trivia Game	USA	Trivia Game. Guess word from 4 pictures.	C
21	Renaissance of Civilizations (文明复)	Strategy Game	China	Strategy Game. Build country and train heroes	C -> Em

	兴)			from ancient civilizations to fight.	
22	FIFA Soccer 2013 EA	Sports Game	Universal	Sports Game. Soccer simulation game.	Em
23	NBA 2K13	Sports Game	Universal	Sports Game. Basketball simulation game.	Em
24	Hay Day	Simulation Game	Universal	Simulation Game. Farm theme simulation game.	C
25	Era of Sail (船长日志)	Simulation Game	China	Simulation Game. Harbor theme simulation game.	Em
26	Zenonia 4	Role Playing Game	Universal	Role Playing Game. Act as hero to fight against enemy and complete a story.	C -> Em
27	Legend of Swords 5(仙剑奇侠传 5)	Role Playing Game	China	Role Playing Game. Act as hero to fight against enemy and complete a story.	C -> Em
28	Asphalt 7	Racing Game	Universal	Racing Game. Drive with fancy cars to win tournament.	C
29	Earn to Die	Racing Game	USA	Racing Game. Drive to smash zombies and earn money.	Em
30	Cut the Rope	Puzzle Game	Universal	Puzzle Game. To solve puzzles by feeding the little monster by cutting ropes	C + Em
31	Find Something (找你妹)	Puzzle Game	China	Puzzle Game. Find the target items to complete goals	C
32	Tap tap Metallica	Music Game	USA	Music Game. Follow the rhythm of the music and tap the buttons	Em
33	Master of Rhythm (乐动达人)	Music Game	China	Music Game. Follow the rhythm of the music and tap the buttons	Em
34	LEGO App4+	Educational Game	USA	Educational Game. Build own trucks to bring cargos to the destination.	Em
35	Chinese Education Committee (天朝教育委员会)	Educational Game	China	Educational Game. Questions and Answers from various subjects	C
36	Pac-Man Lite	Family Game	USA	Family Game. Pac-Man NES simulation	Em
37	Slots™	Casino Game	USA	Casino Game. Slots simulation game.	Em
38	Blackjack Free	Casino Game	USA	Casino Game. Blackjack simulation game.	Em
39	iTunes U	Education	Universal	Education App. Learn public free courses online.	Em
40	Ted	Education	Universal	Education App. Watch video conference about new ideas.	Em
41	Vocabulary Smash (百词斩)	Education	China	Education App. English vocabulary memorizing helper.	C + Em
42	iBooks	Books	Universal	Books App. Ebook reader. Purchase ebooks in apple store or upload ebooks.	C + Em
43	Wattpad	Books	Universal	Books App. Free ebook online reader. Write users'	C + Em

				own book.	
44	Bluefire Reader	Books	Universal	Books App. Simple ebook reader.	C + Em
45	QQ Reader	Books	China	Books App. Simple ebook reader with online purchase feature.	C
46	Box	Business	Universal	Business App. Save and share files in cloud.	C + Em
47	Dropbox	Business	Universal	Business App. Save files in cloud with doc reader feature.	C + Em
48	SplashTop 2	Business	Universal	Business App. Use mobile device to control PC desktop.	C
49	Shell-For iPhone, Style Shopping App Free	Catalogs	USA	Catalogs App. Online iPhone cover store.	Em
50	Jokes	Catalogs	USA	Catalogs App. Jokes sharing app.	Em
51	Guoku (果库)	Catalogs	China	Catalogs App. Online catalogs for Taobao.	Em
52	IMDB Movies & TV	Entertainment	Universal	Entertainment App. Movie and TV rating and details.	C + Em
53	Youku HD (优酷视频)	Entertainment	China	Entertainment App. Online video sharing and playing.	C + Em
54	Bank of American iPad	Financial	USA	Financial App. Personal mobile app for Bank of American users	C
55	Mint.com Personal Finance	Financial	USA	Financial App. Personal financial helper application	C + Em
56	Alipay.com (支付宝) HD	Financial	China	Financial App. Online Payment management application.	Em
57	Healthy Recipes by Spark Recipes for iPad	Food & Drinks	USA	Food & Drinks App. Recipe collection application	Em
58	Pizza Hut HD	Food & Drinks	USA	Food & Drinks App. Business promotion and online ordering for PizzaHut	Em
59	Public Review (大众点评) HD	Food & Drinks	China	Food & Drinks App. Business promotion and user review platform.	C + Em
60	Calorie Counter and Diet Tracker HD	Health & Fitness	USA	Health & Fitness App. Tracking body weight and counting calories	Em
61	Daily Ab Workout Free	Health & Fitness	Universal	Health & Fitness App. body building helper with sit-ups	C
62	Ultimate Diet Theory Free (终极减肥法原理篇)	Health & Fitness	China	Health & Fitness App. Diet helper with suggestions and theories	Em
63	Houzz Interior Design Ideas	Lifestyle	Universal	Lifestyle App. Interior design idea presentation application	C + Em
64	Zulily	Lifestyle	USA	Lifestyle App. Fashion design ideas and products for kids	C
65	Medscape	Medical	USA	Medical App. medical resource and information platform.	C

66	Chunyu Mobile Doctor HD (春雨掌上医生)	Medical	China	Medical App. Medical and health-related adviser.	Em
67	Good Doctor Online (好大夫在线)	Medical	China	Medical App. Online disease information and specific treatment ideas	C
68	Spotify	Music	Universal	Music App. Online music player with social features	C
69	TuneIn Radio	Music	Universal	Music App. Online radio player	C + Em
70	QQ Music HD (QQ 音乐)	Music	China	Music App. Online music player and downloader	C -> Em
71	iGuzheng (爱古筝)	Music	China	Music App. Guzheng playing simulation.	Em
72	Baidu Map	Navigation	China	Navigation App. Mobile map showing locations and directions	C
73	Ship Finder HD	Navigation	USA	Navigation App. Real time ship movement and relevant ship information	C -> Em
74	City Maps 2Go	Navigation	USA	Navigation App. Offline maps and travel guide	C
75	Flipboard	News	Universal	News App. Social news app to customize new selection.	C + Em
76	Feedly	News	Universal	News App. Application to organize read and share RSS feed news.	C -> Em
77	Pocket	News	Universal	News App. Save and organize long internet articles for future reading	C + Em
78	Netease News (网易新闻)	News	China	News App. application for netease.com news.	C
79	Adobe Photoshop Express	Photo & Video	Universal	Photo App. Editing and perfect photos and pictures	C + Em
80	Instagram	Photo & Video	Universal	Photo App. Take pictures and share them instantly.	C + Em
81	PPTV HD	Photo & Video	China	Video App. Online video player.	C
82	Dropbox	Productivity	Universal	Productivity App. Online file storage management and sharing.	C + Em
83	Evernote	Productivity	Universal	Productivity App. Online documents notebook.	C + Em
84	Documents by Readdle	Productivity	Universal	Productivity App. Online documents viewer.	C + Em
85	Bible	Reference	Universal	Reference App. Bible reader.	C + Em
86	Youdao Dictionary HD	Reference	China	Reference App. Mobile dictionary reference	Em
87	Wikipedia Mobile	Reference	Universal	Reference App. Wikipedia Viewer.	Em
88	Facebook	Social	Universal	Social App. Mobile facebook application	C + Em
89	Skype for iPad	Social	Universal	Social App. Mobile Skype application	C + Em
90	Weibo (微博)	Social	China	Social App. Chinese twitter application	C
91	Live Score Addicts	Sports	Universal	Sports App. Online football score and information	C + Em

				viewer	
92	Watch Soccer (看球啦)	Sports	China	Sports App. Online football instant score update and information	C
93	ESPN SportsCenter	Sports	USA	Sports App. ESPN sports news feed viewer	Em
94	Hotel Tonight	Travel	USA	Travel App. Hotel reservation and information	C
95	Elong	Travel	China	Travel App. Hotel and flight reservation and information.	Em -> C
96	Calculator for iPad Free	Utilities	Universal	Utilities App. Calculator simulation	Em
97	Best Flash Light	Utilities	USA	Utilities App. Flashlight simulation	Em
98	Chinese Calendar (万年历)	Utilities	China	Utilities App. Calendar simulation	Em -> C + Em
99	Weather+	Weather	Universal	Weather App. weather report and forecasting	Em
100	Weather Master (天气通)	Weather	China	Weather App. weather report and forecasting	C + Em

*Abbreviation for Maintenance Models

Emergency-Oriented Maintenance Model – **Em**

Event-Oriented Maintenance Model – **Ev**

Constant Maintenance Model – **C**